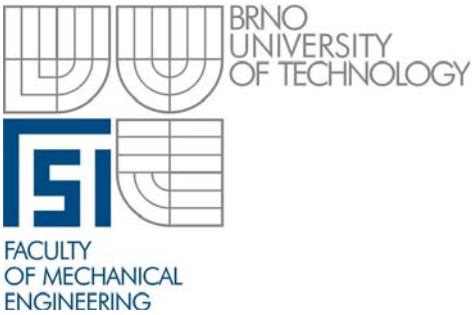


Vysoké učení technické v Brně
Fakulta strojního inženýrství
Ústav automatizace a informatiky

Strukturální metody identifikace objektů pro řízení průmyslového robotu

Dizertační práce

Vypracoval: Ing. Martin Minařík
Školitel: doc. RNDr. Ing. Jiří Šťastný, CSc.

Brno červenec 2009

Abstrakt:

Tato dizertační práce se zabývá využitím strukturálních metod identifikace objektů pro řízení průmyslového robotu.

Nejprve je popsán současný stav znalostí v dané oblasti, tedy celý proces rozpoznávání objektů pomocí metod klasické syntaktické analýzy. Největší nevýhodou je nemožnost rozpoznávat objekty, jejichž digitalizovaný obraz je nějakým způsobem porušen či zkreslen (díky nadměrnému šumu nebo poruchám obrazu), tedy deformován. Pro rozpoznávání deformovaných objektů jsou dále popsány metody využívající strukturálního popisu pro jejich rozpoznávání – metody pro stanovení vzdáleností mezi atributovými popisy obrazů.

Vlastní jádro celé práce spočívá až v kapitole 5, kde jsou popsány deformační gramatiky, jež jsou schopny popsat všechny možné deformace objektu. Komplikací při jejich analýze je nejednoznačnost deformační gramatiky, která snižuje efektivitu analýzy.

Dále je věnován prostor výběru a modifikaci vhodného parseru schopného efektivně analyzovat deformační gramatiku. Popsány jsou tři typy parseru: modifikovaný Earlyho parser, modifikovaný Tomita parser a modifikovaný hybridní LRE(k) parser. Pro Earlyho parser je popsán efektivní způsob jeho implementace.

Nezbytnou součástí rozpoznávání objektů je zajištění invariancí, čímž se tato práce též detailně zabývá. Závěrem jsou uvedeny výsledky popsaných algoritmů (úspěšnost a rychlost rozpoznávání deformovaných objektů) a je popsáno navržené testovací prostředí a algoritmy v něm implementované. V závěru jsou shrnuty zjištěné možnosti deformačních gramatik a jejich výsledky.

Klíčová slova: *strukturální metody, rozpoznávání objektů, syntaktická analýza, deformační gramatika, invariance vůči rotaci, diferenciální primitiva, Earlyho parser, LR(k) parser, Tomita parser, LRE(k) parser, invariance vůči změně měřítka*

Abstract:

This PhD thesis deals with the use of structural methods of objects identification for industrial robots operation.

First, the present state of knowledge in the field is described, i.e. the whole process of objects recognition with the aid of common methods of the syntactic analysis. The main disadvantage of these methods is that is impossible to recognize objects whose digitalized image is corrupted in some ways (due to excessive noise or image disturbances), objects are therefore deformed. Further, other methods for the recognition of deformed objects are described. These methods use structural description of objects for object recognition, i.e. methods which determine the distance between attribute descriptions of images.

The core part of this PhD thesis begins in Chapter 5, where deformation grammars, capable of description of all possible object deformations, are described. The only complication in the analysis is the ambiguity of the deformation grammar, which lowers the effectiveness of the analysis.

Further, PhD thesis deals with the selection and modification of a proper parser, which is able to analyze a deformation grammar effectively. Three parsers are described: the modified Earley parser, the modified Tomita parser and the modified hybrid LRE(k) parser. As for the modified Earley's parser, ways of its effective implementation are described.

One of the necessary parts of the object recognition is providing the invariances, which this PhD thesis covers in detail, too. Finally, the results of described algorithms are mentioned (successfulness and speed of deformed objects recognition) and suggested testing environment and implemented algorithms are described. In conclusion, all determined possibilities of deformation grammars and their results are summarized.

Keywords: *structural method, pattern recognition, syntactic analysis, deformation grammar, rotation invariance, differential primitives, Earley parser, LR(k) parser, Tomita parser, LRE(k) parser, invariance towards change of scale*

Obsah

1 ÚVOD

Důležitou oblastí současných informačních technologií je počítačové vidění (*computer vision*), které se v poslední době prudce rozvíjí. Hlavním úkolem počítačového vidění je rozvíjení metod pro rozpoznání objektů [29], které nacházejí praktické uplatnění v celé řadě úloh, například diagnostické povahy; ze všech možných případů uveďme například průmyslovou robotiku, kde se obvykle pro zpracování obrazu používají metody „Momenty objektu“, „Fourierovy deskriptory“, „Ramena objektu“ a podobně. Vyšší nároky, kladené na metody rozpoznávání obrazu, často i požadavky pro zpracování v reálném čase (řízení průmyslových robotů), vyžadují použití rychlých metod pro zpracování obrazu při zachování vysokého procenta úspěšnosti identifikace jednotlivých objektů.

Metody pro rozpoznávání objektů lze rozdělit podle použitého popisu a způsobu vyhodnocování popisu objektů do dvou základních skupin:

- příznakové metody
- strukturální (syntaktické) metody

První skupina popisuje objekty pomocí číselných charakteristik. Číselný vektor popisující objekt se nazývá *příznakový* vektor a metody využívající jej jsou tudíž známé pod názvem *příznakové* metody. Pokud jsou ovšem důležitým nositelem informací o objektech jejich strukturální charakteristiky, tyto metody nejsou vhodné. Transformací úlohy do příznakového prostoru dochází ke ztrátě strukturálních charakteristik a je obtížné či téměř nemožné je získat zpět. Proto v takových případech je vhodnější popsat objekty metodami druhé skupiny pomocí elementárních popisných vlastností, tak zvaných *primitiv* a relací mezi nimi. [32].

Metody druhé skupiny, u nichž je každý objekt popsán relační strukturou, jsou známé pod názvem *strukturální (nebo syntaktické)* metody. Pokud omezíme popisné relace na jedinou, relaci *„následuje za“*, přechází struktura na posloupnost neboli řetězec symbolů. Každý symbol řetězce odpovídá jednomu popisnému primitivu z množiny primitiv, použitých k popisu. Tuto množinu lze chápat jako abecedu formálního jazyka a řetězce jako slova tohoto formálního jazyka. Úlohu rozpoznávání lze pak popsat jako problém rozhodnout, zda slovo generované danou gramatikou odpovídá řetězci, popisujícímu daný objekt. Zatímco u příznakových metod rozpoznávání je využíván kvantitativní popis předmětů číselnými parametry (*příznakovým vektorem*), u strukturálních metod má vstupní popis kvalitativní charakter odrážející strukturu objektu. Strukturální metody tedy dávají na základě hlubšího teoretického rozboru velmi dobré předpoklady pro úspěšnou identifikaci objektů [19].

Průběh zpracování a rozpoznávání digitalizovaného obrazu lze rozdělit do několika základních kroků:

- *předzpracování obrazu*
- *segmentace obrazu*
- *popis objektů*
- *rozpoznávání objektů*

Při hodnocení algoritmů počítačového rozpoznání obrazu se bere v úvahu jejich úspěšnost klasifikace, podíl nerozpoznaných a špatně klasifikovaných objektů. Důležité jsou i jejich

časové a paměťové požadavky, které dle aplikačního nasazení mohou být klíčové (například řízení průmyslových robotů).

2 CÍLE DIZERTAČNÍ PRÁCE

U klasických strukturálních metod je chybné rozpoznání objektu způsobeno převážně vlivem šumu nebo obrazových poruch ve vstupním obraze. Ačkoliv vhodně použité metody preprocesingu mohou významně zlepšit vlastnosti obrazu a tedy míru úspěšně rozpoznaných objektů, stále se díky nadměrnému šumu nebo poruchám obrazu může vyskytnout zkreslení snímaného tvaru objektů. Klasické metody syntaktické analýzy poté obvykle selžou. Existují dva základní druhy deformace obrazu:

- deformace, které můžeme předpokládat
- náhodné deformace

Mezi deformace, které lze předpokládat, patří deformace rohů objektu nebo deformace krátkých přímých úseků hran objektu. Tyto deformace je vhodné do návrhu analyzátoru zahrnout.

Pro klasifikaci objektů, jejichž tvar je náhodně deformován, je možné použít například metody pro stanovení vzdálenosti mezi atributovými popisy obrazů nebo deformační gramatiky. V dalším textu jsou objekty, jejichž tvar obrazu je náhodně deformován, nazývány z důvodů jednoduššího popisu jako „*deformované objekty*“.

Tato práce je primárně zaměřena na výzkum možností využití strukturálních metod pro rozpoznání náhodně deformovaných objektů se zaměřením na míru úspěšnosti klasifikace. Důležitou součástí této práce je nejen důkladný teoretický rozbor dané problematiky včetně návrhu příslušných algoritmů, ale i jejich implementace v testovacím prostředí, která simulačními experimenty prokáže jejich použitelnost v praxi. Konkrétní cíle práce jsou:

- Analyzovat možnosti rozpoznávání deformovaných objektů pomocí strukturálních metod, konkrétně deformačních gramatik.
- Optimalizovat návrh deformační gramatiky – prozkoumat možnosti návrhu deformační gramatiky s cílem dosažení maximální efektivity analýzy.
- Analyzovat možnosti automatického vytváření gramatiky popisující rozpoznávaný objekt a navrhnout vhodný způsob.
- Analyzovat použitelné parsery a vybrat vhodný typ. Modifikovat jej k umožnění analýzy deformační gramatiky a dále k dosažení dostatečné efektivity analýzy.
- Vhodným způsobem zajistit veškeré potřebné invariance.
- Vytvořit komplexní testovací prostředí obsahující všechny potřebné funkce, které zajistí objektivní zhodnocení analyzované metody.

3 PRŮBĚH ZPRACOVÁNÍ A ROZPOZNÁVÁNÍ DIGITALIZOVANÉHO OBRAZU

Prvním krokem zpracování a rozpoznávání digitalizovaného obrazu je takzvané předzpracování obrazu [33].

3.1 PŘEDZPRACOVÁNÍ OBRAZU – ZÁKLADNÍ OPERACE, FILTROVÁNÍ OBRAZU

Metody předzpracování obrazu slouží k vylepšení obrazu pro potřeby dalšího zpracování. Cílem předzpracování je potlačit šum vzniklý při digitalizaci a přenosu obrazu, odstranit zkreslení dané vlastnostmi snímacího zařízení, případně potlačit nebo zvýraznit jiné rysy důležité z hlediska dalšího zpracování [32].

Základní pixelově nezávislé operace jsou transformace do *stupnice šedi*, *úprava jasu* a *gama korekce*. Velice důležitou částí každého software pro zpracování obrazu je *ekvalizace histogramu*, která vylepšuje kontrast a jejím hlavním cílem je zisk jednotného histogramu. Tyto operace mohou být aplikovány na celý obraz nebo jen na jeho část. Jejich podrobný popis lze najít například v [13].

Dále je zde specifická skupina transformací nazývaná filtrace, která transformuje hodnoty jasu vstupního obrazu do jiných hodnot jasu za účelem zvýraznění nebo potlačení některých charakteristik obrazu. Často se provádí vyhlazování, tedy odstranění vysokých frekvencí obrazu. Nejpoužívanější typy filtrů (pro danou aplikační oblast) jsou:

- filtrování průměrováním
- medián filtry
- filtrování posuvným průměrováním
- filtrování Gaussovým filtrem
- filtrování pomocí Fourierovy transformace

Tyto filtry jsou detailně popsány například v [31]. Zmiňované operace nejsou nezbytné, obecně mají negativní vliv na rychlost zpracování, ale mohou výrazně vylepšit celkové výsledky (celková rychlost, úspěšnost klasifikace).

3.2 SEGMENTACE

Segmentace je další důležitá operace, jedná se o dekompozici obrazu na jednotlivé objekty a pozadí. Patrně nejjednodušší metodou segmentace je prahování (*thresholding*) [7]. Je založena na skutečnosti, že mnoho objektů nebo oblastí obrazu má konstantní odrazivost nebo pohltivost povrchu. K oddělení objektů od pozadí lze pak využít určenou jasovou konstantu *(práh)*. Vzhledem k jednoduchosti výpočtu je prahování nejrychlejší segmentační metodou. Prahování je transformace vstupního obrazu *f* na výstupní binární obraz *g* daná vztahem:

$$g(i,j) = 1 \quad \text{pro} \quad f(i,j) > T$$
$$g(i,j) = 0 \quad \text{pro} \quad f(i,j) \leq T$$

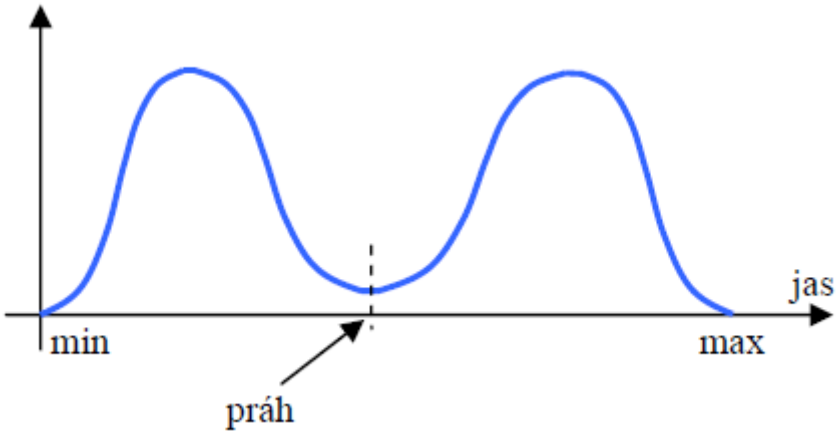
kde: T je předem určená konstanta *(práh)*, $g(i,j)=1$ pro obrazové elementy náležející po segmentaci objektům a $g(i,j)=0$ pro elementy pozadí (nebo naopak). Správné určení prahu má zásadní vliv na výsledek prahování. Hodnotu prahu lze určovat buď interaktivně ve spolupráci s uživatelem, nebo automaticky. Na obr. 1 (převzato z [32]) je ilustrována volba správné hodnoty prahu.

Obr. 1 Prahování vhodným prahem, nízkým prahem a vysokým prahem

Globální prahování nemusí vždy vykazovat požadované segmentační výsledky. To může být způsobeno nerovnoměrným osvětlením scény a podobně. Tehdy je vhodné použít prahování s proměnným prahem (*Adaptive Thresholding*), podrobněji v [7], kdy je hodnota prahu určována podle lokálních vlastností obrazu.

3.2.1 Metody určování prahu

Existuje více metod pro určování prahu, obvykle se používá některá z následujících, podrobněji v [36, strana 38]:



Obr. 2 Bimodální histogram s naznačeným prahem

- *Procentní prahování*. Nejjednodušší situace nastává, je-li předem známa vlastnost, kterou má obraz po segmentaci mít. Práh se pak určuje tak, aby tuto vlastnost obraz po segmentaci měl. Například pokud víme, že objekty pokrývají *1/p* plochy obrazu, pak je

snadné určit (na základě histogramu) takovou hodnotu prahu *T*, aby právě *1/p* plochy mělo úrovně jasu menší než *T*. Ale tato vlastnost většinou není známa, a proto se využívají jiné metody.

- *Analýza tvaru histogramu*. Jsou-li v obraze objekty (přibližně stejného jasu) jasově odlišné od pozadí, je histogram jasu dvouvrcholový (*bimodální*). Jeden vrchol odpovídá četností obrazových elementů pozadí, druhý četnosti elementů objektů. Ilustrováno na obr. 2. Práh je potom vhodné volit jako hodnotu jasu, jejíž četnost je minimem ležícím mezi těmito dvěma maximy. V případě, že je histogram *multimodální* (vícevrcholový), lze určit více hodnot prahů. Segmentace se pak provádí podle více prahů.

3.3 DETEKCE HRAN

Hrany v obrazu nesou informace o obrazu. Definují, kde objekty jsou, jejich tvar, velikost a též i něco o jejich textuře. Detekce hran v obrazu je úzce spojena s preprocesingem, jedná se vlastně o proces, který určuje, zda bod obrazu je element hrany či element pozadí. Hrany se v obrazu nachází v místech, kde se náhle mění jas, hranové detektory tedy v obrazu hledají ostré změny v obrazové funkci.

Hranové detektory lze rozdělit do dvou kategorií. První detekuje známé objekty v obrazu, druhá detekuje objekty bez předchozí znalosti zpracovávané scény. Výhoda druhé kategorie je její nezávislost na zpracovávaném obrazu. Pro detekci hran se používají například metody: Dilatace – eroze, gradientní metody prvního řádu (Sobel, Roberts, Prewitt, izotopický operátor, operátor kompas…), gradientní metody druhého řádu (Laplace, operátor průchodu nulou, Canny detektor…), Wavelet transformace a další. Srovnání hranových detektorů je možno najít například v [30]. Je nezbytné zvolit metodu, která pro dané použití poskytne dostatečné výsledky a přitom je co nejrychlejší. Výsledky hranových detektorů lze pozitivně ovlivnit použitím vhodných filtrů.

3.4 POPIS OBJEKTŮ

Další důležitou fází rozpoznávání obrazu je popis nalezených oblastí, což není zcela triviální. Je nezbytné nalézt popis, jenž umožní zajistit invariance vůči posunu, rotaci a případně i změně měřítka.

- U syntaktického přístupu je obraz reprezentován řetězcem, tvořeným sledem primitiv obrazu.
- Je-li objekt popsán hranicí, která je uzavřená, lze snadno nalézt primitiva. Řetězcem těchto primitiv je následně definován tvar objektu.
- Objekt je rozdělen na konečný počet elementárních částí – primitiv (přímka, úsečka, křivka, oblouk a podobně).
- Každému primitivu je přiřazen určitý symbol, který se nazývá terminál.
- Řetězec terminálních symbolů je generován gramatikou [9] a reprezentuje daný objekt.
- Vztahy mezi primitivy jsou popsány nejčastěji jednorozměrnými relacemi - příkladem jednorozměrné relace je relace uspořádání, která představuje zřetězení terminálů.

3.4.1 Návrh primitiv

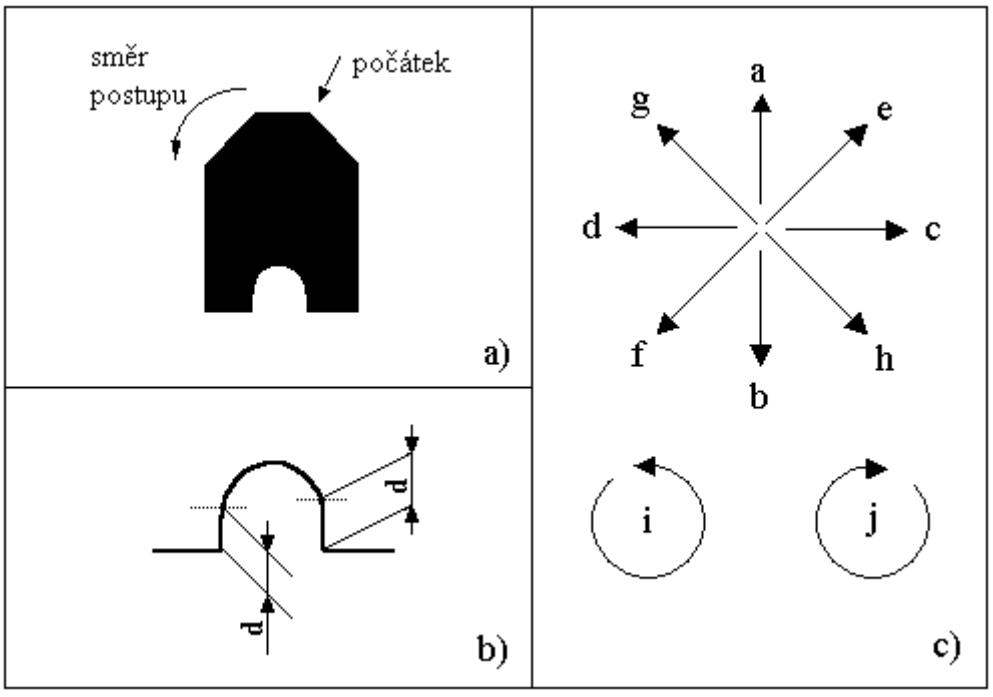
Primitiva jsou základním stavebním prvkem obrazu u strukturálních metod rozpoznávání. Pro obrazy charakterizované hranicí, jsou vhodnými primitivy části čar. Například úsečka může být charakterizována svým počátkem a koncem, délkou, případně úhlem. Podobně mohou být charakterizovány i části křivek. Návrh primitiv závisí na řešené aplikaci. Při jejich návrhu je vhodné zvážit možnost jejich snadného rozpoznání [32]. Obecně platí:

- Primitiva musí být snadno rozpoznatelná.
- Primitiva musí poskytovat kompaktní a postačující popis obrazů pomocí specifikovaných vztahů.

Za pomoci složitějších primitiv dostaneme jednodušší strukturální popis objektu, což vede k jednodušší gramatice pro popis objektu, ale důsledkem je zvýšení složitosti při hledání takových primitiv v obraze. Naopak jednodušší primitiva vedou sice ke složitější gramatice, jejich výhodou však je jednodušší identifikace v obraze.

3.4.2 Příklad popisu objektu a primitiv

Na obr. 3 (převzato z [32]) jsou primitiva *a, b, c, d, e, f, g, h* zvolena jako úsečky, primitiva *i, j* jako kladný a záporný oblouk. Je též potřeba určit počátek a směr postupu popisu objektu. Osmiokolím a dvěma oblouky (obr. 3c) lze dostatečně popsat všechny objekty. Jednodušší varianta primitiv může být obdobně čtyřokolí, které ale postačuje pouze v omezené oblasti objektů. Složitější primitiva závisí na popisovaných objektech a zvyšují složitost popisu objektů, ale ve výsledku mohou vést k významnému zefektivnění analýzy scény.



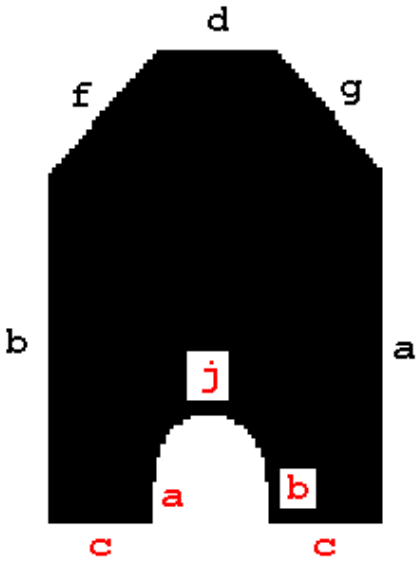
Obr. 3 Domek s detailem a navržená primitiva

V případě, že postupujeme od vyznačeného počátečního bodu (obr. 3a) proti směru chodu hodinových ručiček, dostaneme řetězec:

d f b c a j b c a g

Vlivem špatné kvality vstupního obrazu se může stát, že se nebudou detekovat krátké rovné úseky (obr. 3b) a tím se změní i řetězec objektu (obr. 4, převzato z [32]).

- d f b c a j b c a g
- d f b c j b c a g
- d f b c a j c a g
- d f b c j c a g



Obr. 4 Objekt s popisem alternativními řetězci

Pro tento případ je možné upravit gramatiku tak, aby generovala i alternativní řetězce obr. 5 (převzato z [32]).

Gramatika generující řetězec d f b c a j b c a g	Gramatika generující řetězce d f b c j b c a g d f b c a j c a g d f b c j c a g
S → dA A → fB B → bC C → cD D → aE E → jB E → g	S → dA A → fB B → bC C → cD C → cE D → aE E → jB E → jC E → g

Obr. 5 Úprava gramatiky pro alternativní řetězce

Tato forma přístupu je, jak je dále podrobněji popsáno, vhodná pro předpokládané chyby (deformace obrazu) při identifikaci daného objektu. V případě náhodných deformací je možné použít klasifikaci pomocí algoritmů pro zjištění vzdálenosti mezi atributovými popisy obrazů (viz kapitola 4), nebo deformační gramatiky (viz kapitola 5).

3.5 ROZPOZNÁVÁNÍ OBJEKTŮ (PATTERN RECOGNITION)

Rozpoznávání předmětů (klasifikace) spočívá v zařazování předmětů do tříd. *Třída* je množina objektů, jejichž atributy mají z hlediska klasifikace společné rysy. *Předmět* je fyzikální objekt, který v počítačovém vidění představuje nejčastěji určitou oblast segmentovaného obrazu [32].

Samotnou činnost klasifikace vykonává *klasifikátor*. Klasifikátor nerozhoduje o třídě předmětu podle předmětu skutečného, nýbrž podle jeho vjemu. Tento vjem se nazývá *obraz* (předmětu). Pro jednoduchost pod pojmem *objekt* je v textu míněn segmentovaný obraz jednoho předmětu s detekovanými hranami.

Pro vlastní klasifikaci objektů existuje řada různých metod, podrobněji například v [29]. Každá z nich má svoje výhody i nevýhody, proto volba té nejvhodnější metody je závislá na konkrétní aplikaci. Například u neuronových sítí je oproti klasickým algoritmům hlavní rozdíl v tom, že u umělých neuronových sítí [32], [15] nemusíme znát algoritmus řešení daného problému. Postačuje znalost určitého počtu příkladů a jejich řešení. Dále neuronové sítě vykazují vynikající výsledky, avšak s jednou nevýhodou. Příznakový vektor je u nich tvořen tak, že ignoruje vnitřní částí objektů a předměty jsou rozpoznávány jen podle své vnější hrany.

3.6 ROZPOZNÁVÁNÍ OBJEKTŮ POMOCÍ STRUKTURÁLNÍCH METOD

V příznakových metodách rozpoznávání je využíván kvantitativní popis objektů číselnými parametry – příznakovým vektorem. U syntaktického přístupu je obraz reprezentován řetězcem, tvořeným sledem primitiv obrazu a polohových vztahů mezi nimi, tedy vstupní popis má kvalitativní charakter odrážející strukturu objektu [32]. Máme-li objekt popsán hranicí, která je uzavřená, můžeme snadno nalézt primitiva. Těmito primitivy je pak definován tvar objektu. Takto zpracovaný obraz se dále používá v rozpoznávacích metodách. Řetězec terminálních symbolů (*slovo*) může reprezentovat daný obraz. Množina slov, popisujících obrazy jedné třídy tvoří jazyk této třídy. Jazyk, pokud je konečný, může být definován množinou jeho slov. Jednotlivá slova jazyka jsou generována takzvanými přepisovacími pravidly gramatiky.

Jeden ze způsobů, jak rozpoznat strukturu daného neznámého obrazu, spočívá v porovnání jeho strukturální reprezentace ve formě řetězce s reprezentacemi vzorových obrazů jednotlivých tříd. Takový způsob je nezbytný například v úlohách, kdy počet tréninkových vzorů je nedostatečný pro odvození gramatik, nebo když každý obraz může být považován za prototyp třídy obrazů. Tyto případy lze též řešit pomocí algoritmů pro zjištění vzdálenosti mezi atributovými popisy obrazů. Tento způsob je jednoduchý a rychlý. Je-li však třeba pro rozpoznání úplný popis obrazu, je nezbytná syntaktická analýza.

Úkolem syntaktického rozpoznávání obrazu je určit, zda analyzovaný obraz reprezentovaný slovem odpovídá slovu dané gramatiky, tedy zda gramatika může tento obraz generovat. Obraz je reprezentován řetězcem jazyka, který je generován danou gramatikou.

3.6.1 Typy gramatik

Některé jazyky lze generovat gramatikami, které mají přepisovací pravidla speciálního typu, tedy nevyužívají všech možností nabízených obecnou definicí gramatiky [9]. Gramatiky lze rozdělit podle různých hledisek do více typů, jedno z nejpoužívanějších rozdělení je takzvaná Chomského hierarchie gramatik:

- Gramatika typu 0 – neomezená gramatika (omezená pouze podstatou věci) – je uspořádaná čtveřice $G = (N, T, P, S)$ kde: N – množina neterminálů, T – množina terminálů (abeceda), S – počáteční symbol $(S \in N)$, P – konečná množina přepisovacích pravidel tvaru:

$$\alpha \rightarrow \beta, \text{ kde } \alpha \in (N \cup T)^+, \beta \in (N \cup T)^*$$

- Gramatika typu 1 – kontextová gramatika - je gramatika $G = (N, T, P, S)$, jejíž přepisovací pravidla mají tvar:

$$\gamma\alpha\delta \rightarrow \gamma\beta\delta, \text{ kde } \alpha \in (N \cup T)^+; \beta, \gamma, \delta \in (N \cup T)^*, |\gamma| + |\delta| > 0$$

  γ, δ je kontext. Jedinou výjimkou může být pravidlo $S \rightarrow e$, jehož výskyt znamená, že se S nesmí objevit na pravé straně žádného přepisovacího pravidla z množiny P. Jazyk typu 1 (kontextový jazyk) je jazyk, generovaný kontextovou gramatikou.
- Gramatika typu 2 – bezkontextová gramatika - je gramatika $G = (N, T, P, S)$, jejíž přepisovací pravidla mají tvar:

$$\alpha \rightarrow \beta, \text{ kde } \alpha \in N, \beta \in (N \cup T)^*$$

  Bezkontextová gramatika se nazývá *nevypouštějící,* jestliže neobsahuje žádné přepisovací pravidlo typu: $\alpha \rightarrow e$, kde e označuje prázdný řetězec. Ke každé bezkontextové gramatice G existuje *nevypouštějící bezkontextová gramatika* G_1 taková, že platí: $L(G_1) = L(G) - \{e\}$. Jazyk typu 2 (bezkontextový jazyk) je jazyk, generovaný bezkontextovou gramatikou.
- Gramatika typu 3 – regulární gramatika - je gramatika $G = (N, T, P, S)$, jejíž přepisovací pravidla mají tvar:

$$A \rightarrow wA \mid w, \text{ kde } A \in N, w \in T^*$$

  Jazyk typu 3 (regulární jazyk) je jazyk, generovaný regulární gramatikou. Tyto jazyky jsou rozpoznatelné konečnými automaty. Slovo generované regulární gramatikou G je rozpoznatelné konečným automatem A, jestliže platí: $L(G) = L(A)$. Regulární gramatika může být levá nebo pravá.

Jednotlivá slova jazyka jsou generována tak zvanými přepisovacími pravidly gramatik vhodného typu. Pomocí regulárních gramatik lze obvykle řešit většinu případů z řešené aplikační oblasti.

3.6.2 Přepisovací systémy

Tvorba slov podle určité gramatiky se děje postupným přepisováním řetězců v souladu s jejími pravidly. Přepisovací systém je uspořádaná dvojice $R = (T, P)$, kde:

- T – konečná abeceda
- P – konečná množina přepisovacích pravidel

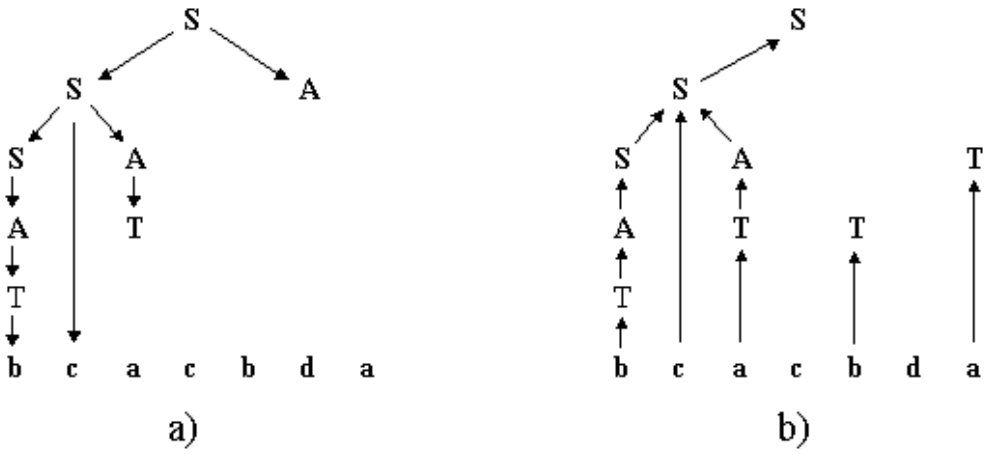
Každé přepisovací pravidlo je uspořádaná dvojice (α, β), kde $\alpha, \beta \in T^*$. Přepisovací pravidla se zapisují ve tvaru $\alpha \rightarrow \beta$.

3.7 SYNTAKTICKÁ ANALÝZA

Syntaktická analýza je proces, který rozhoduje o náležitosti řetězce do jazyka generovaného gramatikou (tedy rozpoznání objektu), pro rozpoznání:

- Regulární gramatiky - stačí zkonstruovat odpovídající deterministický konečný automat, který má obvykle snadnou softwarovou i hardwarovou realizaci.
- Bezkontextová gramatika - obecně nedeterministický zásobníkový automat.
- Kontextová gramatika – „rozumné“ syntaktické analýzy lze dosáhnout pomocí bezkontextové gramatiky s řízeným přepisováním.

Existují dva základní způsoby (podrobněji například v [35], [4]) syntaktické analýzy. Jedná se o analýzu shora-dolů (*top down*) a analýzu zdola-nahoru (*bottom up*), viz obr. 6. Při syntaktické analýze zpočátku známe jen řetězec a počáteční symbol gramatiky. Mezi těmito dvěma údaji tvoříme invertovanou stromovou strukturu.



Obr. 6 Analýza a) shora-dolů a b) zdola-nahoru

3.7.1 Syntaktická analýza zdola – nahoru (*bottom – up parsing*)

Postupujeme od analyzovaného řetězce směrem k počátečnímu symbolu. Analýza začíná s prázdným zásobníkem. V případě úspěšného přijetí řetězce zůstane v zásobníku pouze

počáteční symbol, příkladem je Cocke–Younger–Kasami algoritmus [28], který zaručuje, že čas potřebný k syntaktické analýze je úměrný pouze třetí mocnině délky řetězce.

3.7.2 Syntaktická analýza shora – dolů (*top – down parsing*)

Vycházíme z počátečního symbolu a snažíme se vygenerovat analyzovaný řetězec. Doposud vygenerovaný řetězec ukládáme do zásobníku. Vždy, když se na vrcholu zásobníku objeví terminální symbol, porovná se s aktuálním vstupním symbolem analyzovaného řetězce. V případě souhlasu se terminální symbol z vrcholu zásobníku odstraní. V opačném případě se vrátíme tak daleko, kde lze zvolit jiné pravidlo (třeba za pomocí backtrackingu). Příkladem je Earlyho parser (popsán dále), který provádí všechny možné způsoby analýzy současně takovým způsobem, že často může zkombinovat již získané částečné výsledky. Čas potřebný k syntaktické analýze je úměrný třetí mocnině délky řetězce. Není-li gramatika víceznačná, je čas potřebný k analýze úměrný dokonce jen druhé mocnině délky řetězce. Modifikace tohoto algoritmu byla použita v řešeném simulačním prostředí.

3.8 EARLYHO PARSER

Earlyho parser (*Earley Parser,* podrobnější popis například v [21], [27], [36], [4]) je dynamický algoritmus provádějící syntaktickou analýzu shora dolů. Algoritmus je pojmenován po svém objeviteli – Jay Earley. Earlyho parsery jsou zvláště zajímavé, protože dokáží zpracovávat bezkontextové gramatiky. Jsou zvláště efektivní, pokud jsou pravidla analyzované gramatiky zapsána pomocí *levé rekurze* [20].

V následujícím popisu algoritmu α, β, a γ reprezentuje jakýkoliv řetězec terminálů/neterminálů (včetně prázdného řetězce); *X, Y, a Z* reprezentuje jeden neterminál a jeden terminál je reprezentován pomocí symbolu *a*. V textu je použita též Earlyho tečková notace: je dáno pravidlo $X \rightarrow \alpha\beta$, zápis $X \rightarrow \alpha \bullet \beta$ reprezentuje stav, ve kterém α již bylo zpracováno a β je očekáváno ke zpracování.

Pro každou vstupní pozici (která reprezentuje pozici mezi znaky), parser generuje množinu stavů. Každý stav je dvojice $(X \rightarrow \alpha \bullet \beta, i)$, sestávající z právě zpracovávaného pravidla $(X \rightarrow \alpha\ \beta)$, momentální pozice zpracování v pravidle (reprezentované tečkou) a pozice *i* ve vstupu, na které zpracování tohoto pravidla začalo. Earlyho originální algoritmus obsahoval též předpověď stavů, další výzkum ukázal pouze malý přínos efektivitě parseru a následně byla předpověď vypuštěna z většiny obvyklých implementací.

Množina stavů vstupní pozice *k* se označuje *S(k)*. Parser je na počátku naplněn množinou *S(0)*, spočívající pouze v pravidlech nejvyšší úrovně. Parser pracuje iterativně pomocí třech operací: predikce, scanování a kompletace.

- Predikce: Pro každý stav v *S(k)* tvaru $(X \rightarrow \alpha \bullet Y\ \beta, j)$ (kde *j* je původní úroveň stavu), přidej $(Y \rightarrow \bullet\ \gamma, k)$ do *S(k)* pro každé pravidlo s *Y* na levé straně.
- Scanování: Pokud *a* je další symbol ve vstupním řetězci, pro každý stav v *S(k)* tvaru $(X \rightarrow \alpha \bullet a\ \beta, j)$, přidej $(X \rightarrow \alpha\ a \bullet \beta, j)$ do *S(k+1)*.
- Kompletace: Pro každý stav v *S(k)* tvaru $(X \rightarrow \gamma \bullet, j)$ najdi stavy *S(j)* tvaru $(Y \rightarrow \alpha \bullet X\ \beta, i)$ a přidej je $(Y \rightarrow \alpha\ X \bullet \beta, i)$ do *S(k)*.

Tyto kroky se opakují, dokud není možno žádný další stav přidat do množiny. Množina je obvykle implementována jako seznam stavů ke zpracování (přesto se daný stav musí v seznamu objevit pouze jednou) a vykonávání odpovídajících operací závisí na druhu stavu.

3.9 ALGORITMUS SYNTAKTICKÉ ANALÝZY

Algoritmus syntaktické analýzy [32] rozpoznávající objekty na dané scéně včetně zajištění invariance vůči rotaci lze popsat takto:

1. Jestliže nejsou analyzovány všechny řetězce, načti nový řetězec a pokračuj krokem 2, jinak pokračuj krokem 7.
2. Proveď syntaktickou analýzu pro vybranou třídu.
3. Jestliže řetězec patří do jazyka gramatiky vybrané třídy, pokračuj krokem 6.
4. Jestliže počet rotování řetězce je menší než délka řetězce, rotuj řetězec a pokračuj krokem 2, jinak pokračuj krokem 5. Rotování řetězce znamená přesunutí terminálního symbolu z poslední pozice na první.
5. Jestliže počet otočení předmětu je menší než 360/úhlový krok, otoč předmět o zadaný úhlový krok a pokračuj krokem 2. Rotování předmětu znamená pootočení předmětu o zadaný úhel a tím získání jiného řetězce.
6. Zapiš výsledek a pokračuj krokem 1.
7. Vypiš zprávu o rozpoznání objektu.

Pokud potřebujeme klasifikovat N objektů, musíme vytvořit N tříd, N gramatik pro ně a odpovídající jazyky $L(G_1), L(G_2), ..., L(G_N)$. Například pokud gramatika G_x generuje slova obsahující pouze jeden symbol „*b*“, budou všechna slova obsahující pouze jeden symbol „*b*“ patřit do třídy X náležející této gramatice. Objekty obsahující více než jeden symbol „*b*“ budou dále analyzovány zbývajícími gramatikami. V případě, že nebude nalezena žádná gramatika odpovídající danému řetězci, bude objekt potlačen.

Při návrhu syntaktického analyzátoru je vhodné předpokládat náhodné vlivy, například deformace obrazu. V případě primitiv odpovídajícím jednotlivým segmentům hran je tato gramatika velice citlivá na (i velmi malé) chyby v detekci hran.

3.10 ZAJIŠTĚNÍ INVARIANCÍ

Strukturální popis objektů pomocí primitiv je invariantní vůči posunu, ale není invariantní vůči rotaci a vůči volbě bodu počátku popisu objektu. Pro praktickou použitelnost této metody je tedy nezbytné tyto invariance zajistit. V algoritmu uvedeném v předchozí kapitole je zajištěna invariance vůči rotaci v bodu 5. algoritmu pomocí otáčení objektu a tímto získáním nového řetězce. Invariance vůči volbě bodu počátku popisu je zajištěna v bodu 4. algoritmu rotací řetězce.

Efektivnější způsob zajištění invariance vůči rotaci pro tuto metodu je normalizace předkládaných slov, tedy předkládaná slova – řetězce vždy objekt popisují ve stejné poloze. Jedna z možností normalizace používaná u strukturálních metod vychází z myšlenky, že každému terminálnímu symbolu (primitivu) lze přiřadit jedno číslo, při zde používaném osmi – okolí a dvěma obloukům postačí číslice 0 – 9. Normalizovaný řetězec získáme jako minimum všech možných kombinací čísel vzniklých rotací o jeden znak řetězce popisující daný objekt. Tento způsob je však nepoužitelný u případů, kde jednomu objektu mohou

odpovídat slova různých délek, tedy je typicky nepoužitelný pro zajištění invariance u rozpoznávání deformovaných objektů.

4 ROZPOZNÁVÁNÍ DEFORMOVANÝCH OBJEKTŮ

Z předchozího textu je patrné, že rozpoznávání nedeformovaných objektů pomocí strukturálních metod je bezproblémové, nabízí výbornou rychlost a stoprocentní úspěšnost klasifikace, avšak u náhodně deformovaných objektů je téměř nemožné.

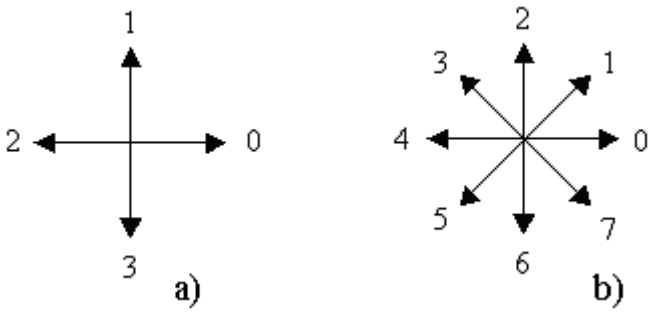
4.1 METODY PRO ZJIŠTĚNÍ VZDÁLENOSTÍ MEZI ATRIBUTOVÝMI POPISY OBRAZŮ

Tyto metody nepatří přímo mezi strukturální metody, ale pouze využívají strukturálního popisu objektů ve formě řetězce. Někdy se o nich hovoří jako o metodách meziřetězcových vzdáleností, což není jejich zcela správný název, ale zároveň velice dobře vystihuje hlavní myšlenku těchto metod. Tato vzdálenost udává míru podobnosti dvou řetězců, z nichž jeden odpovídá analyzovanému objektu a druhý obrazu třídy a lze ji využít pro rozpoznávání objektů. Pro stanovení této meziřetězcové vzdálenosti lze použít nejrůznější metody ke stanovení vzdálenosti mezi atributovými popisy obrazů [32]. Reprezentace analyzovaného obrazu a reprezentace vzorových obrazů jednotlivých tříd se mohou lišit nejen syntakticky, ale také sémanticky. Nejprve je nutno určit reprezentaci objektu pomocí řetězce.

4.1.1 Řetězcové kódy

Freemanovy řetězcové kódy (*chain codes*) [8] se používají k popisu hranic objektů. Hranice je určena počátečním bodem a posloupností symbolů, odpovídajících úsečkám jednotkové délky v několika předem stanovených směrech. K popisu hranice se používají dvě varianty řetězcových kódů:

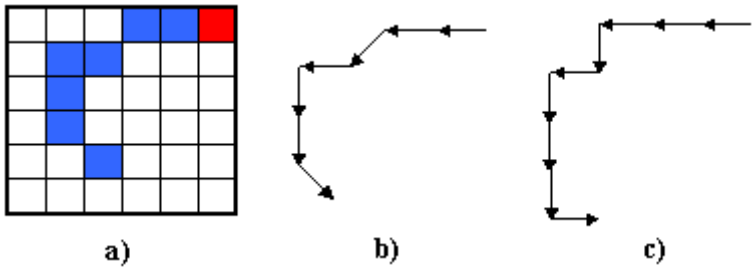
- 4 směry po 90° viz obr. 7a) a obr. 8a), c), převzato z [32]
- 8 směrů po 45° viz obr. 7b) a obr. 8a), b), převzato z [32]



Obr. 7 Směry řetězcových kódů

- Řetězcový kód pro 4 směry: 2,2,2,3,2,3,3,3,0
- Zápis pomocí exponentů pro 4 směry: $2^3,3^1,2^1,3^3,0$

- Řetězcový kód pro 8 směrů: 4,4,5,4,6,6,7
- Zápis pomocí exponentů pro 8 směrů: $4^2,5^1,4^1,6^2,7^1$

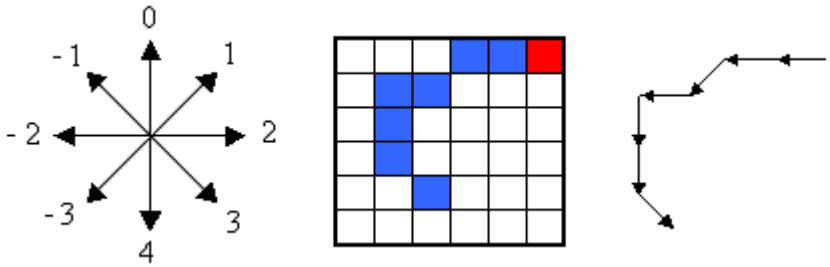


Obr. 8 Příklad pro 4 a 8 směrů

Z uvedených zápisů je patrné, že řetězcový kód pro 8 směrů je kratší než řetězcový kód pro 4 směry. Tento typ řetězcového kódu splňuje invarianci vůči posunutí, avšak nesplňuje požadavek invariance vůči rotaci. Řetězcový kód pro osm směrů odpovídá osmi – okolí popisovaném v kapitole 3.4.2.

4.1.2 Diferenciální řetězcové kódy

Diferenciální řetězcové kódy představují variantu *Freemanova řetězcového kódu*. Diferenciální řetězcový kód je invariantní vůči posunutí i rotaci. Diferenciální řetězcový kód odpovídá diferenciálním primitivám používaným v kapitole 7.2



Obr. 9 Diferenciální řetězcový kód

- Zápis diferenciálního kódu pro hranu z obr. 9:
- 0, 0,-1, 1,-2, 0,-1

4.1.3 Hammingova vzdálenost

Hammingova vzdálenost $H_d(s,t)$ [10], [23] patří mezi základní a nejčastěji používané metody, protože je jednoduchá a snadno se realizuje. Je definována pouze pro řetězce o stejných délkách. Pro dva řetězce s a t je Hammingova vzdálenost $H_d(s,t)$ definována jako počet míst, ve kterých se tyto dva řetězce liší, neboli počet znaků, v nichž se tyto dva řetězce liší. Celková vzdálenost je pak dána součtem hodnot těchto rozdílů:

$$H = \sum_{i=1}^{N} |a_i - b_i|$$

Ze vztahu je patrné, že vzdálenost je dána součtem všech rozdílných bodů dvou obrazců. Tato metoda se používá pro porovnání binárních vektorů. Nejčastěji to bývají černobílé obrazce, kde 0 reprezentuje bílou a 1 reprezentuje černou barvu. Ze znalosti tohoto faktu lze ještě zjednodušit výpočet použitím binární operace XOR:

$$|a_i - b_i| = a_i \ XOR \ b_i$$

Výhodou této metody je její rychlost, avšak značnou nevýhodou je použitelnost pouze pro porovnávání dvou řetězců o stejných délkách.

Důsledkem nevýhody, spočívající v omezení použitelnosti této metody pouze na řetězce o stejných délkách, je praktická nepoužitelnost Hammingovy metody pro zjištění vzdálenosti mezi atributovými popisy obrazů. Popis obrazu může být různě deformován vlivem nejrůznějších poruch (například šumu, špatného nasnímání obrazu, volbou filtrů…), a proto málokdy získáme popis obrazu se stejnou délkou řetězce, jakým je popsán vzor, se kterým je získaný popis srovnáván.

4.1.4 Euklidova vzdálenost

Jednou z nejznámějších metrik je *Euklidova vzdálenost* [10], [34]. Předpokládejme kartézský souřadný systém, ve kterém jsou dva vektory *A* a *B*. Pak obecně pro libovolnou dimenzi prostoru platí:

$$E = \sqrt{\sum_{i=1}^{N} \left(A(i) - B(i) \right)^2}$$

kde *N* je dimenze prostoru.

4.1.5 Levenshteinova vzdálenost

Levenshteinova vzdálenost $L_d(s, t)$ [12], [10] je definována jako míra podobnosti mezi dvěma řetězci *s* a *t*. Levenshteinova vzdálenost $L_d(s, t)$ je počet vložení, smazání a substitucí znaků potřebných k transformaci řetězce *s* na řetězec *t*. Levenshteinova metoda poskytuje rozšířenou reprezentaci vzdálenosti mezi dvěma porovnávanými řetězci.

4.1.6 Příklady Levenshteinovy vzdálenosti

- Příklad 1. Řetězce jsou identické
  s: d d f f f b b b c c c a a c c
  t: d d f f f b b b c c c a a c c
  Levenshteinova vzdálenost $L_d(s, t) = 0$

- Příklad 2. Vložení znaku
  s: d d f f f b b b c c c a a c c
  t: d d f f f b b b c c c - a c c

vložení znaku ‘a’, potom Levenshteinova vzdálenost $L_d(s, t) = 1$

- Příklad 3. Smazání znaku
  s: d d f f f b b b c c c a a c c
  t: d d f f f b b b c c c a a c c
  smazání znaku ‘a’, potom Levenshteinova vzdálenost $L_d(s, t) = 1$

- Příklad 4. Substituce znaku
  s: d d f f f b b b c c c a a c c
  t: d d f f f b b b c c c c a c c
  substituce ‘c’ -> ‘a’, potom Levenshteinova vzdálenost $L_d(s, t) = 1$

4.1.7 Needleman-Wunsch vzdálenost

Metoda Needleman-Wunsch [10] vypočítává vzdálenost mezi řetězci použitím matic cen za provedenou operaci. Do výsledku se promítnou ceny, které mohou mít pro každou operaci různé hodnoty. Povolené operace vložení, smazání a substituce mohou mít tedy každá různou cenu. Pro každou operaci existuje jedna matice cen. Uvnitř matice cen můžeme definovat pro každou buňku zvláštní ohodnocení.

Metoda Needleman-Wunsch poskytuje podobnou reprezentaci vzdálenosti mezi dvěma porovnávanými řetězci jako Levenshteinova vzdálenost. Tuto metodu opět můžeme použít i pro nestejně dlouhé řetězce. Hlavní rozdíl ve srovnání s Levenshteinovou metodou je v možnosti ohodnocení různých operací cenami. Tento rozdíl však neposkytuje pro uvažované aplikace žádnou výraznou výhodu. Vzhledem k tomu, že časová náročnost této metody je přibližně stejná jako u Levenshteinovy metody, je též metoda Needleman-Wunsch vhodná pro toto použití.

4.1.8 Bloková vzdálenost (Manhattan)

Bloková vzdálenost [10] je zjednodušená verze Euklidovy vzdálenosti. Opět se provádí měření Euklidovy vzdálenosti, ale s vynecháním mocnin:

$$M = \sum_{i=1}^{N} |A(i) - B(i)|$$

Tato vzdálenost již nebude minimální, ale bude splňovat kritéria metriky, tedy bude udávat míru vzdálenosti. Tato metoda je kompromisem mezi rychlostí a přesností.

4.1.9 Daemerauova vzdálenost

Daemerauova vzdálenost $D_d(s, t)$ [10], [9] je téměř identická s Levenshteinovou vzdáleností s tím rozdílem, že umožňuje kromě operací vložení, smazání a substituce ještě operaci výměny dvou sousedních znaků. Daemerauova vzdálenost $D_d(s, t)$ je tedy definována jako počet vložení, smazání, substitucí a výměn dvou sousedních znaků, potřebných k transformaci řetězce *s* na řetězec *t*.

Damerauova vzdálenost rovněž poskytuje rozšířenou reprezentaci vzdálenosti mezi dvěma porovnávanými řetězci. Výhodou této metody je možnost použití i pro nestejně dlouhé řetězce, stejně jako u Levenshteinovy metody. Její odlišnost od Levenshteinovy metody (přidaná operace výměny dvou sousedních znaků pro dosažení rovnosti řetězců) však pro použití při zjišťování vzdáleností mezi atributovými popisy obrazů není potřebná (spíše by mohla být výhodou pro jiné aplikace) a navíc může zvýšit čas provádění výpočtu. Tento čas však může být pro určité aplikace zanedbatelný, takže lze říci, že se tato metoda hodí pro zjištění vzdálenosti mezi atributovými popisy obrazů.

4.1.10 Jaccardova vzdálenost

Jaccardova vzdálenost J_d [10] je definována pouze pro řetězce o stejných délkách, podobně jako Hammingova vzdálenost. Pro dva řetězce *s* a *t* je Jaccardova vzdálenost $J_d(s,t)$ definována jako poměr počtu identických znaků na stejných pozicích v řetězcích *s* a *t* k počtu různých znaků na stejných pozicích řetězců *s* a *t*.

Jaccardova vzdálenost poskytuje podobné výsledky jako Hammingova vzdálenost, a sice základní reprezentaci vzdálenosti mezi dvěma porovnávanými řetězci. Stejně jako u Hammingovy vzdálenosti je její výhodou rychlost, avšak její nevýhodou je použitelnost pouze pro porovnávání dvou řetězců o stejných délkách.

Důsledkem nevýhody, spočívající v omezení použitelnosti této metody pouze na řetězce o stejných délkách, je praktická nepoužitelnost Jaccardovy metody (stejně jako Hammingovy vzdálenosti) pro zjištění vzdálenosti mezi atributovými popisy obrazů.

4.1.11 Minkowského vzdálenost

Minkowského vzdálenost $M_d(s,t,power)$ [3] je geometrickou vzdáleností mezi dvěma řetězci bod po bodu (znak po znaku). Argument *power* může být použit pro generování Manhattanské vzdálenosti (pokud je *power = 1*) nebo Euklidovské vzdálenosti (pokud je *power = 2*). Tato metoda zahrnuje stejně jako Levenshteinova vzdálenost operace vložení, smazání a substituce znaků.

Minkowského metodu můžeme opět použít i pro nestejně dlouhé řetězce. Zajímavé možnosti poskytuje dynamicky počítané ohodnocení ceny za operaci.

Protože pomocí Minkowského metody můžeme porovnávat i řetězce nestejných délek, mohla by být tato metoda vhodná pro zjištění vzdálenosti mezi atributovými popisy obrazů. Tato metoda má však nevhodnou vlastnost vzhledem k uvažované aplikační oblasti. Zahrnuje dynamicky počítané ohodnocení ceny za operaci podle vzdálenosti porovnávaných znaků (jedná se o vzdálenost v ASCII tabulce), které může chovat nedeterministicky – výsledné vypočítané hodnoty vzdáleností mohou mít značné výkyvy.

4.2 ROZPOZNÁVÁNÍ OBJEKTŮ NA PRINCIPU MINIMÁLNÍ VZDÁLENOSTI

Předpokládejme, že v obrazovém prostoru je zadáno R bodů $v_1, v_2, \cdots, v_R$, které se nazývají *etalony*, neboli vzorové obrazy tříd $\omega_1, \cdots, \omega_r$. Klasifikátor podle minima

vzdálenosti zařadí klasifikovaný obraz x do té třídy, jejíž etalon má od bodu x nejmenší vzdálenost. Přítomnost ke třídě ω_r je určena vztahem:

$$x : \omega_r \Leftrightarrow |v_r - x| = \min |v_s - x_s|$$

4.3 ANALÝZA METOD PRO ZJIŠTĚNÍ VZDÁLENOSTI MEZI ATRIBUTOVÝMI POPISY OBRAZŮ

Z dostupných metod pro zjištění vzdálenosti mezi atributovými popisy obrazů byly analyzovány metody: Hammingova vzdálenost $H_d(s,t)$, Levenshteinova vzdálenost $L_d(s,t)$, Damerauova vzdálenost $D_d(s,t)$, Jaccardova vzdálenost J_d, Minkowského vzdálenost $M_d(s,t,power)$ a metoda Needleman-Wunsch.

Nevhodné pro uvažovanou aplikační oblast jsou metody Hammingova a Jaccardova, protože nejsou schopny porovnávat řetězce nestejných délek, což pro zjištění vzdálenosti mezi atributovými popisy obrazů je nepostradatelná podmínka (rozlišené popisné řetězce nalezených vzorků se mohou značně lišit od řetězcových popisů etalonů vlivem nejrůznějších poruch obrazu, které vzniknou v procesu předzpracování nebo snímání obrazu).

U Minkowského metody je použití pro aplikaci na zjištění vzdálenosti mezi atributovými popisy obrazů velmi problematické, neboť dynamicky počítané ohodnocení ceny za operaci podle vzdálenosti porovnávaných znaků (vzdálenost v ASCII tabulce) může zapříčinit značné výkyvy výsledné vypočítané hodnoty vzdáleností.

Zbývající tři metody Levenshtein, Damerau a Needleman-Wunsch jsou použitelné pro zjištění vzdálenosti mezi atributovými popisy obrazů. U metody Damerau však přidaná operace výměny dvou sousedních znaků pro uvažované aplikace zbytečně komplikuje implementaci algoritmu. U Needleman-Wunsch metody je možnost ohodnocení různých operací cenami obtížně využitelná, nezpůsobuje však velkou komplikaci algoritmu.

Na základě analýzy [32] uvedených metod byly jako nevhodnější pro zjištění vzdálenosti mezi atributovými popisy obrazů vybrány metody Levenshteinova a Needleman-Wunsch.

Žádný z uvedených algoritmů není invariantní vůči natočení, proto každému musíme předkládat vždy jeden z popisných řetězců tolikrát, kolik znaků představuje jeho délka, při čemž se v každém kroku provede rotace řetězce o jeden znak. Pak se vybere nejkratší zjištěná řetězcová vzdálenost. Tím se časová složitost každého z algoritmů zvýší v závislosti na délce rotujícího řetězce.

4.4 VLASTNÍ ALGORITMUS

Algoritmus pro rozpoznávání obrazu pomocí vyhodnocování vzdáleností mezi atributově popsanými objekty [32] probíhá ve dvou hlavních fázích:

1. výpočet vzdáleností
2. vyhodnocení vzdáleností

1. Výpočet vzdáleností:

V této fázi se porovnávají rozpoznané objekty s etalony. Výsledkem této fáze jsou všechny zjištěné vzdálenosti, které existují mezi všemi rozpoznanými objekty (včetně všech jejich rotací) a všemi etalony. Všechny vypočítané vzdálenosti nese každý objekt s sebou i s informací, která vzdálenost přísluší kterému objektu (ve druhé fázi se provádí vyhodnocení na základě právě těchto vypočítaných vzdáleností).

1. Jestliže nejsou analyzovány všechny objekty, načti další objekt a pokračuj krokem 2, jinak pokračuj krokem 6.
2. Jestliže nejsou analyzovány všechny etalony, načti další etalon a pokračuj krokem 3, jinak pokračuj krokem 1.
3. Jestliže nejsou vyčerpány všechny rotace řetězcové reprezentace objektu, rotuj objekt a pokračuj krokem 4, jinak pokračuj krokem 2.
4. Vypočítej řetězcovou vzdálenost mezi rotovanou řetězcovou reprezentací objektu a řetězcovou reprezentací etalonu podle zadaného algoritmu pro zjištění vzdálenosti mezi dvěma řetězci a pokračuj krokem 5.
5. Zapiš zjištěnou vzdálenost a pokračuj krokem 1.
6. Proveď druhou fázi – vyhodnocení vzdáleností.

2. Vyhodnocení vzdáleností:

V této fázi se vyhodnotí „míra příslušnosti“ každého etalonu ke každému objektu podle vypočítaných vzdáleností z předcházející fáze. „Míra příslušnosti“ je dána uživatelsky nastavitelnými parametry *hloubka* a *percentil* (pojmy definované v [32]).

1. Jestliže nejsou analyzovány všechny objekty, načti další objekt a pokračuj krokem 2, jinak pokračuj krokem 7.
2. Seřaď všechny vzdálenosti objektu (zjištěné z první fáze) do pole vzdáleností objektu a pokračuj krokem 3.
3. Jestliže nejsou analyzovány všechny etalony „příslušející” objektu, načti další etalon a pokračuj krokem 4, jinak pokračuj krokem 1.
4. Vypočítej *percentil* pro objekt a jeho „příslušející” etalon.
5. Pokud objektu příslušející zjištěná vzdálenost (zjištěné z první fáze) je menší než vzdálenost z pole vzdáleností objektu (krok 2) s indexem *hloubka* minimálních vzdáleností (zadává se nejlépe v aplikaci realizující tento algoritmus) a zároveň platí, že aktuálně vypočítaný *percentil* (krok 4) je větší než zadaný *percentil* (zadává se též nejlépe v aplikaci realizující tento algoritmus), pak objekt vyhovuje etalonu, zapiš objektu příznak.
6. Pokračuj krokem 1.
7. Konec rozpoznávání, vypiš zprávu o rozpoznání – objekty, které mají nastaven příznak (z kroku 5) a jim odpovídající etalony, popřípadě zjištěné vzdálenosti, *percentil*.

4.5 ZHODNOCENÍ METOD PRO ZJIŠTĚNÍ VZDÁLENOSTI MEZI ATRIBUTOVÝMI POPISY OBRAZŮ

Výsledky těchto metod jsou uvedeny např. v [32]. Informovanost těchto metod není nejvyšší, jedná se pouze o „nějakou“ metriku, proto tyto metody mohou produkovat nerozpoznané a chybně rozpoznané objekty. Pokud jsou parametry (hloubka a percentil)

metod pro zjištění vzdáleností mezi atributovými popisy obrazů správně nastaveny, tyto metody nabízejí relativně dobrou míru úspěšně identifikovaných objektů při dosažení výborné rychlosti klasifikace. Avšak chybné rozpoznání objektu může nastat.

5 SYNTAKTICKÁ ANALÝZA S OPRAVOU CHYB

Předchozí kapitola pojednávající o metodách využívajících strukturálního popisu objektu k jejich rozpoznávání ukázala, že je možné využít strukturálních metod i pro rozpoznávání náhodně deformovaných objektů, ovšem úspěšnost popsaných metod není vždy nejvyšší. Snahou této práce je tuto úspěšnost co nejvíce zvýšit, případně zcela zamezit chybnému rozpoznání objektu a tedy najít metodu vhodnou i pro „přesné“ rozpoznávání náhodně deformovaných objektů. Dosáhneme toho důsledným využitím všech dostupných znalostí o cílech klasifikace, tedy vlastní gramatiky popisující objekt. Oproti metodám popsaným v předchozí kapitole je tedy tento přístup informovanější, více v [17], [18].

Pokud budeme provádět klasickou syntaktickou analýzu řetězce popisujícího náhodně deformovaný objekt nějaké třídy, patrně nebude do dané třídy klasifikován, nebo jen zcela „náhodou“. Řešením je rozšíření původní gramatiky o chybová – deformační pravidla zahrnující všechny možné náhodné deformace objektu. Původní gramatika je regulární nebo bezkontextová, rozšířená deformační gramatika je vždy bezkontextová a navíc víceznačná, tedy její syntaktická analýza bude složitější. Úloha se poté změní na úlohu nalezení nedeformovaného řetězce, jehož vzdálenost je od analyzovaného minimální [35].

5.1 KONSTRUKCE ROZŠÍŘENÉ OBECNÉ DEFORMAČNÍ GRAMATIKY

Rozšířená deformační gramatika má za úkol spolehlivě generovat všechny možné deformace objektů, jež mohou nastat.

Vstup: Bezkontextová nebo regulární gramatika $G = (N, T, P, S)$.
Výstup: Rozšířená gramatika $G' = (N', T', P', S')$, kde P' je množina váhových pravidel.

Krok 1: Jedná se o vytvoření terminálních a neterminálních symbolů rozšířené gramatiky. Rozšířená gramatika má být schopna generovat stejné objekty jako gramatika původní, její množina terminálu tedy obsahuje stejné symboly jako množina terminálních symbolů původní gramatiky a též někdy může obsahovat navíc nějaké terminály popisující deformace.

Množina neterminálních symbolů rozšířené gramatiky oproti původní obsahuje navíc deformační neterminály, což jsou neterminály generující všechny možné deformace původních terminálů.

$$N' = N \cup \{S'\} \cup \{E_B \mid b \in T\}$$
$$T \subseteq T'$$

Krok 2: Do rozšířené gramatiky vložíme pravidla popisující deformace původních terminálů v pravidlech původní gramatiky.

Je-li v P pravidlo:

$$A \to \alpha_0 b_1 \alpha_1 b_2 ... \alpha_{m-1} b_m \alpha_m ; m \geq 0; \alpha_l \in N' \wedge b_i \in \Sigma; i = 1,2,...,m; l = 0,1,...,m$$

Potom do P' přidej pravidlo:

$$A \to \alpha_0 E_{b1} \alpha_1 E_{b2} ... \alpha_{m-1} E_{bm} \alpha_m$$ s váhou 0.

Krok 3:

- Do *P'* přidej následující pravidla s váhou odpovídající zvolené vzdálenosti, viz tab. 1; použita Levenshteinova vzdálenost – L, váhová Levenshteinova vzdálenost – w, váhová metrika – W.
- Pravidla typu b) d) e) f) jsou nazývána deformačními pravidly.
- S' v pravidle typu a) je počáteční symbol nové rozšířené gramatiky, pravidla typu b) popisují deformace počátečního symbolu, pravidla typu c) přepisují nově přidané neterminály na terminály, pravidla typu d) popisují jejich náhodné deformace, pravidla typu e) obsahují přepis na prázdný symbol a pravidla typu f) popisují všechny možné náhodné deformace hran objektu.

pravidlo:	L	w	W	pro:
a) $S' \to S$	0	0	0	-
b) $S' \to Sa$	1	w_I	$I'(a)$	$a \in \Sigma'$
c) $E_a \to a$	0	0	0	$a \in \Sigma$
d) $E_a \to b$	1	w_S	$S(a,b)$	$a \in \Sigma, b \in \Sigma', a \neq b$
e) $E_a \to \lambda$	1	w_D	$D(a)$	$a \in \Sigma$
f) $E_a \to bE_a$	1	w_I	$I(a,b)$	$a \in \Sigma, b \in \Sigma'$

Tab. 1 Pravidla rozšířené deformační gramatiky

S rozšířenou gramatikou *G'* pracuje *Syntaktický analyzátor s opravou chyb* [35], který vyhledává takovou deformaci vstupního řetězce, která je spojena s nejmenším součtem vah chybových (deformačních) pravidel. *G'* je víceznačná gramatika.

Problém je tedy najít vhodnou, dostatečně rychlou metodu, schopnou analyzovat víceznačnou gramatiku a akumulovat váhy deformačních pravidel. Parserů splňujících tato kritéria je více, v této práci byly analyzovány následující tři metody:

- Modifikovaný Earlyho parser
- Modifikovaný Tomita parser
- Modifikovaný hybridní LRE(k) algoritmus (hybrid mezi Earlyho parserem a LR(k) parserem)

V navrženém testovacím prostředí byl pro syntaktickou analýzu s opravou chyb použit modifikovaný Earlyho algoritmus, který oproti původní variantě navíc akumuluje příslušné váhy pravidel, použité při derivaci deformovaného řetězce podle gramatiky *G'*.

Deformační gramatika sestrojená podle algoritmu uvedeného v předchozí kapitole je maximálně univerzální, obsahuje deformace původních pravidel, všechny možné deformace každého terminálu v jakýkoliv jiný terminál nebo kombinaci terminálů, deformace počátečního symbolu, deformace terminálu na prázdný řetězec. Výhoda takto navržené

deformační gramatiky je její univerzální použitelnost pro obsáhlou skupinu problémů. Ovšem při rozpoznávání deformovaných objektů si obvykle vystačíme s jednodušší gramatikou, která ačkoliv je méně obecná, poskytuje díky menšímu počtu pravidel efektivnější (rychlejší) rozpoznávání objektů.

5.2 DEFORMAČNÍ GRAMATIKA PRO ROZPOZNÁVÁNÍ OBJEKTŮ

Jak již bylo řečeno, výše popsaná deformační gramatika je pro rozpoznávání objektů zbytečně obecná a je možné ji zjednodušit. Podobně jako u obecné deformační gramatiky nejprve vytvoříme množiny terminálních a neterminálních symbolů, jak je popsáno v předchozí kapitole (krok 1), a vložíme pravidla popisující deformace původních terminálů v pravidlech původní gramatiky (krok 2).

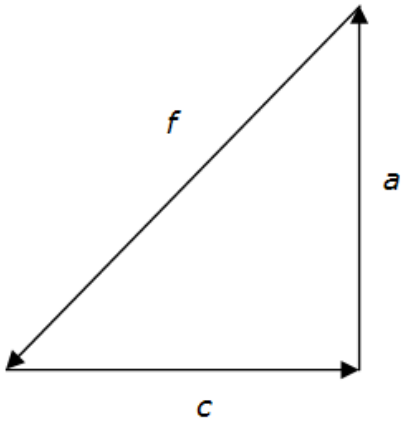
Změny nastávají až v kroku 3, kde deformace počátečního symbolu (3a, 3b) jsou zbytečné. Krok 3e (e-pravidla) se při některých typech zápisu gramatiky popisující objekt mohou používat, v některých typech zápisu jsou zbytečná a též dle použitého parseru a způsobu, jakým parser zpracovává e-pravidla, mohou být též zbytečná.

5.3 DEFORMAČNÍ GRAMATIKA PRO ROZPOZNÁVÁNÍ DEFORMOVANÝCH OBJEKTŮ A JEJÍ VARIANTY

Vždy se snažíme najít pro daný problém co nejvhodnější gramatiku k dosažení maximální efektivity a jednoduchosti při její syntaktické analýze. U klasických strukturálních metod rozpoznávání objektů se obvykle snažíme využít regulárních gramatik, jejichž analýza je jednoduchá a efektivní. Možných způsobů zápisu gramatiky (pro jednoduchost zvolen trojúhelník) je více:

Typ 1:
$G = (N,\sum,P,S)$
$N = (S,A,B)$
$\sum = (a,c,f)$
$P = \{$
$S \rightarrow fA$
$A \rightarrow cB$
$B \rightarrow a\}$

Tato regulární gramatika generuje jednoduchý trojúhelník *„fca“* (viz obr. 10). Její možnosti jsou tímto ale vyčerpány. Podobným způsobem je možné napsat gramatiku pro popis jakéhokoliv objektu. Avšak v případě objektu, pro jehož popis by bylo třeba značné množství znaků, tento způsob přestává být výhodný z důvodů příliš velkého počtu pravidel a formálních problémů při jejich zápise (nutnost pojmenování neterminálních symbolů více znaky). Sofistikovanější způsob zapsání gramatiky generující různé „velikosti“ trojúhelníku pomocí regulární gramatiky je:



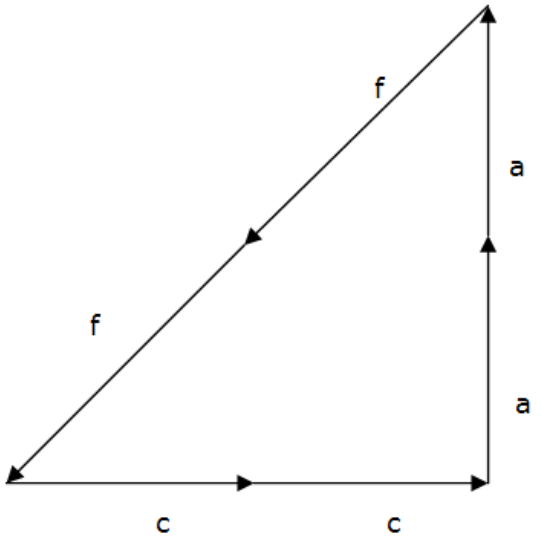
Obr. 10 Typ 1 gramatiky pro popis trojúhelníku

Typ 2:
G = (N, ∑,P,S)

N = (S,A,B,C)

∑ = (a,c,f)
P = {
S -> A
A -> fB|fA
B -> cC|cB
C -> a|aC}

Regulární gramatika zapsaná tímto způsobem je schopná vygenerovat různě velké trojúhelníky (*fcb, ffccbb*…). Tento zápis ovšem skýtá úskalí, pravidla jsou zapsána pomocí takzvané levé rekurze [4], kterou řada parserů není schopná zpracovat. Levou rekurzi lze odstranit použitím e-pravidel, které opět ovšem některým parserům působí obtíže.



Obr. 11 Gramatika typu 2 pro popis trojúhelníku

Pokud se nebudeme omezovat na regulární gramatiky, získáme další možnosti, jak zapsat gramatiku popisující objekt, například při použití bezkontextové gramatiky lze trojúhelník „*ffccaa*“ (obr. 11) zapsat třeba takto:

Typ 3:
G = (N, ∑,P,S)
N = (S,A,B,C)
∑ = (a,c,f)
P = {
S -> FFCCAA
F -> f
C -> c
A -> a

V tab. 2 jsou srovnány tyto tři způsoby zápisu gramatiky generující popis objektu a na první pohled je patrný velký rozdíl mezi nimi, spočívající v rozdílném počtu generovaných stavů a tedy následně rychlosti. Proto je nezbytné podle řešeného problému vybrat nejvhodnější variantu zápisu gramatiky a následně i vhodný parser pro její analýzu. Údaje o době analýzy jsou pouze orientační.

U námi řešeného problému je nutno analyzovat deformační gramatiky, které nejsou jednoznačné, a proto je nutno vybrat parser, který je toho schopen. Takové parsery obvykle jsou schopny zároveň analyzovat bezkontextové gramatiky, a tak v řešeném projektu je používán třetí typ zápisu gramatiky popisující objekt.

Gramatika pro trojúhelník *(fffffffffccccccccaaaaaaaaa)* zapsaná způsobem			
	Typ 1	Typ 2	Typ 3
Počet položek v SListC	424	2303	945
Počet položek v SListP	2657	2051	705
Celkový počet položek	3081	4012	1650
čas [s]	4.062	9.672	0.890
Gramatika pro čtverec *(dddbbbcccaaa)* zapsaná způsobem			
Počet položek v SListC	90	275	178
Počet položek v SListP	387	512	248
Celkový počet položek	477	787	426
čas [s]	0.078	0.234	0.047

Tab. 2 Srovnání různých typů gramatiky pro popis objektu

5.4 MODIFIKOVANÝ EARLYHO ALGORITMUS

Modifikace Earlyho parseru k zajištění korektní analýzy deformačních gramatik spočívá v akumulaci vah pravidel v průběhu analýzy, podrobněji v [35].

Vstup algoritmu:

- Rozšířená gramatika G'
- Vstupní řetězec $w = b_1 b_2 .. b_m$

Výstup algoritmu:

- Seznamy I_0, I_1,..., I_m pro řetězec ***w***
- Vzdálenost ***d*** vstupního řetězce od řetězce obrazu

Krok 1: Zkonstruujeme I_0.

- Pro každé pravidlo $S' \to \alpha \in P'$ přidáme do I_0 položku $[S' \to \bullet\alpha, 0, x]$
- Prováděj tak dlouho, dokud lze do I_0 přidávat položky. Pokud je v I_0 položka $[A \to \bullet B\beta, 0, y]$, přidej pro všechna pravidla $B \overset{z}{\to} \gamma$ položku $[B \to \bullet\gamma, 0, z]$ do I_0.

$$\alpha \to \beta, \text{ kde } \alpha \in (N \cup T)^+, \beta \in (N \cup T)^*$$

Krok 2: Opakujeme pro j = 1, 2,..., m

- Pro každou položku v I_{j-1} ve tvaru $[B \to \alpha \bullet a\beta, i, x]$ takovou, že $a = b_j$, přidej do I_j položku $[B \to \alpha a \bullet \beta, i, x]$, dále prováděj B a C tak dlouho, dokud lze do I_j přidat nějakou položku.
- Jestliže je položka $[A \to \alpha \bullet, i, x]$ v I_j a položka $[B \to \beta \bullet A\gamma, k, y]$ v I_i, pak:
  - Existuje-li již položka ve tvaru $[B \to \beta A \bullet \gamma, k, z]$ v I_j , pak pokud $\boldsymbol{x + y < z}$ nahradíme u této položky hodnotu **z** hodnotou $\boldsymbol{x + y}$.
  - Neexistuje-li, přidáme novou položku $[B \to \beta A \bullet \gamma, k, x + y]$.
- Pro každou položku typu $[A \to \alpha \bullet B\beta, i, x]$ v I_j přidáme pro všechna pravidla $B \overset{z}{\to} \gamma$ položky $[B \to \bullet\gamma, j, z]$

Krok 3:

- Pokud je položka $[S' \to \alpha \bullet, 0, x]$ v I_m, pak řetězec *w* je přijat s váhou vzdáleností *x*. Řetězec *w* (respektive jeho derivační strom) získáme vynecháním všech deformačních pravidel z derivace řetězce *w*.

5.5 EFEKTIVNÍ IMPLEMENTACE EARLEYHO ALGORITMU

Popis Earlyho algoritmu je uveden v celé řadě publikací a článků…například ho lze najít v [21], [27], [36], [4], ovšem snad vždy jsou popsány pouze jeho hlavní principy, jeho podrobný popis a případné poznámky, jak ho efektivně implementovat, zcela chybí. Tato kapitola se podrobně zabývá jeho efektivní implementací s uvážením jeho dalšího použití pro analýzu nejednoznačné deformační gramatiky.

Breath-first, bottom-up parsery, mezi které lze Earlyho parser zařadit, jsou oblíbené díky svým vlastnostem, pracují on-line, dokáží zpracovat levou rekurzi a lze je upravit i ke zpracování e-pravidel.

Obecně analýza prohledáváním do šířky je značně neefektivní a prohledávání je potřeba nějakým způsobem omezit. Bylo nalezeno množství metod k realizaci toho omezení prohledávání do šířky, povětšinou na úkor obecnosti gramatiky, jež jsou tyto metody schopny zpracovat. Jednu z nich, která omezí větvení při prohledávání do šířky do rozumných mezí při zachování plné obecnosti zpracované gramatiky, objevil v roce 1970 Jay Earley.

Earlyho parser je obvykle zařazován do kategorie parserů provádějící analýzu shora dolů (*top-down*), ale ve skutečnosti tento parser dosahuje svých výborných výsledků též omezeným prohledáváním do šířky s rozpoznáváním zdola nahoru (*bottom-up*). Dále při uvážení jeho hlavních vlastností, schopnosti zpracovat levou rekurzi a požadavků na speciální opatření pro zpracování e-pravidel někdo preferuje jeho zařazení jako shora dolů (*bottom-up*) metoda. Lze též najít určitou podobnost mezi Earlyho algoritmem a CYK algoritmem, podrobněji například v [28].

5.5.1 Earlyho parser bez předpovědi (*look-ahead*)

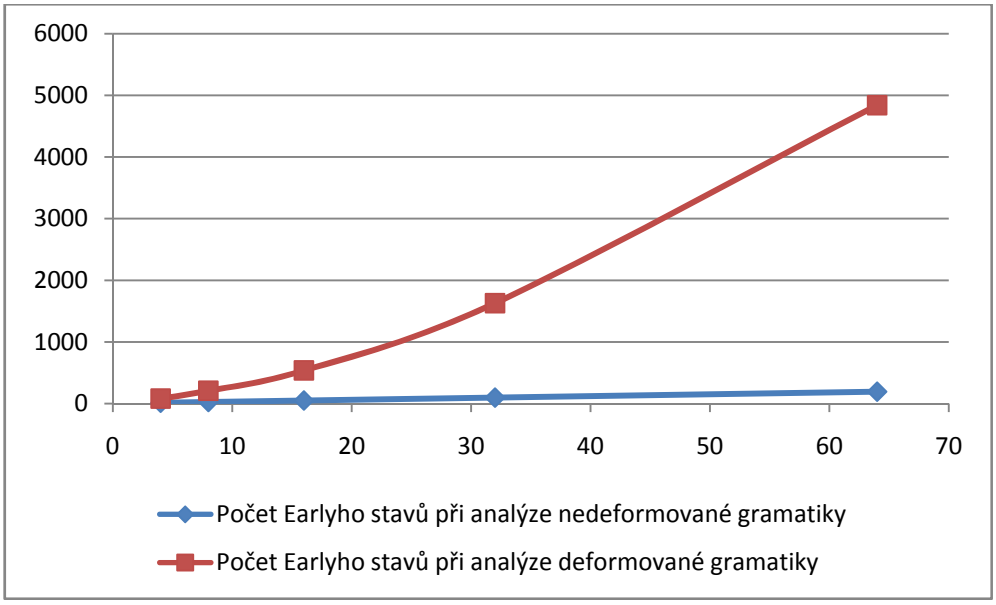
Pokud se podíváme na analýzu nějakého řetězce pomocí prohledávání do šířky, uvidíme, že velké množství provedených operací (redukce) jsou zcela zbytečné. Earley si povšimnul společné vlastnosti těchto zbytečných redukcí (které jsou nekompatibilní s přístupem shora dolů), že je nemožné derivovat je z počátečního symbolu. Proto navrhnul metodu, která redukce omezí pouze na ty, které se mohou derivovat z počátečního symbolu. Výsledný parser má časovou složitost n^3 pro vstupní řetězec délky n, oproti C^n u prohledávání do šířky.

Při analýze nejednoznačné gramatiky se počet generovaných Earlyho stavů s rostoucí délkou řetězce významně zvětšuje, pro ilustraci některé hodnoty jsou uvedeny v tab. 3 a závislost je zobrazena v grafu 1. U analýzy klasické gramatiky je závislost lineární a u analýzy deformační gramatiky je závislost exponenciální.

Délka slova	Počet Earlyho stavů	Počet Earlyho stavů DFG
4	13	80
8	25	207
16	49	537
32	97	1629
64	193	4841

Tab. 3 Závislost počtu generovaných stavů na délce slova

Jeden z hlavních důvodů nižší efektivity analýzy deformační gramatiky je, že algoritmus tráví značné množství času prohledáváním jednotlivých úrovní seznamu stavů. Proto je vlastní implementace toho seznamu stavů pro celkovou rychlost algoritmu zcela klíčová. Například datová struktura TList z Delphi (indexovaný lineární spojový seznam) je svojí rychlosti zcela nevyhovující, výrazně lepší je dynamické pole. Dále je výhodný oddělený přístup k jednotlivým úrovním seznamu z důvodu efektivnějšího prohledávání seznamu stavů. Poznamenejme, že je nutno ponechávat i předchozí úrovně seznamu přístupné během celé analýzy, protože se neustále používají.



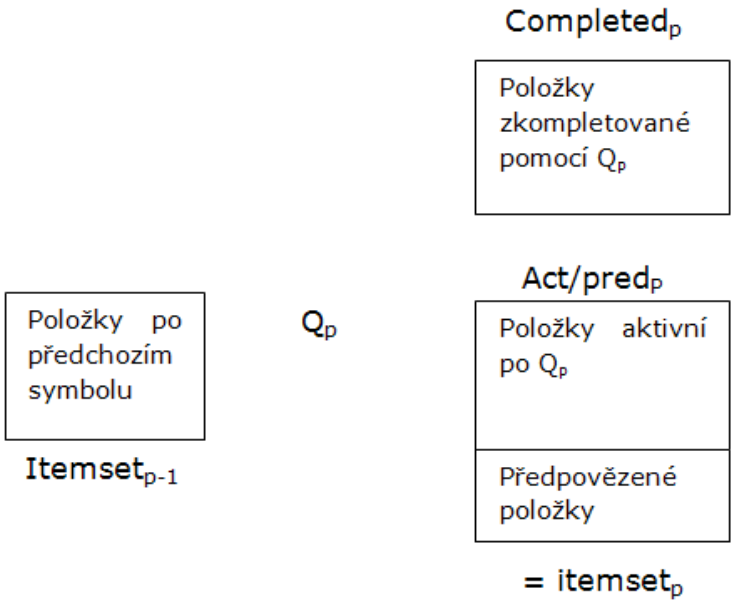
Graf 1 Závislost počtu generovaných stavů na délce slova

Konstrukce jedné úrovně seznamu stavů probíhá ve třech fázích. Dvě odpovídají neomezenému algoritmu, kde byly nazývány „*shift*" a „*reduce*", zde jsou nazývany „*scan*" a „*complete*". Třetí operace je nová a je vztažená k *top-down* komponentě algoritmu, nazývá se „*predictor*".

Earlyho stavy v seznamu stavů se dají rozdělit do dvou druhů, první jsou již zkompletované stavy (znak „*" Earlyho tečkové notace je až na konci stavu) a druhý, nezkompletované stavy (znak „*" Earlyho tečkové notace není na konci stavu). Predikce prohledává nezkompletované stavy, kompletace prohledává již zkompletované stavy a scanování prohledává obecně oba druhy stavů. Prohledávání seznamu stavů při provádění některé z operací můžeme tedy dále značně zefektivnit rozdělení každé úrovně seznamu stavu do dvou částí, jedna pro již zkompletované stavy a druhá pro ještě nezkompletované stavy. Výsledkem každé operace je nějaký nový stav, který se správně zařadí do příslušného seznamu (zkompletovaných/nezkompletovaných) stavů.

Tedy Scanner, Completer a Predictor zpracovávají čtyři seznamy stavů pro každý znak vstupního řetězce, jak je ilustrováno na obr. 12, kde jsou pro vstupní symbol Q_p znázorněny čtyři seznamy stavů:

- $Itemset_{p-1}$ – seznam, který obsahuje pouze stavy dostupné před zpracováním Q_p
- $Completed_p$ – seznam stavů, které byly zkompletovány díky Q_p
- $Active_p$ – seznam, který obsahuje nezkompletované stavy po průchodu Q_p
- $Predicted_p$ – seznam nově předpovězených stavů



Obr. 12 Earlyho seznamy stavů pro jeden vstupní symbol

Seznamy c*ompleted*$_p$, *active*$_p$ a *predicted*$_p$ dohromady formují *itemset*$_p$; vnitřní dělení je naznačeno čarou a pro vlastní analýzu není důležité. Na počátku *itemset*$_{p-1}$ je naplněn (jako výsledek zpracování Q_{p-1}) a ostatní seznamy jsou prázdné. Konstrukce setu *itemset*$_0$ je speciální (viz popis Earlyho algoritmu v kapitole 3.8). Pokud *completed* set pro poslední symbol vstupního řetězce obsahuje položku *(S -> ... *, 0),* tedy položku obsahující celý vstupní řetězec a redukující ji na počáteční symbol, byla analýza úspěšná a vstupní řetězec lze vygenerovat danou gramatikou.

5.5.2 Příklad analýzy řetězce pomocí Earlyho algoritmu

Uveďme si příklad analýzy gramatiky z obr. 13 a vstupního řetězce „*a-a+a*“.

```
S -> E
E -> EQF
E -> F
F -> a
Q -> +
Q -> -
```

Obr. 13 Testovací gramatika A

Celá analýza řetězce je v následující tabulce:

S(0): *a-a+a				
(1)	S->*E	0	Act$_0$	Z počátečního pravidla
(2)	E->*EQF	0	Pred$_0$	Předpovězeno z (0)(1)
(3)	E->*F	0	Pred$_0$	Předpovězeno z (0)(2)
(4)	F->*a	0	Pred$_0$	Předpovězeno z (0)(3)
S(1): a*-a+a				
(1)	F->a*	0	Comp$_1$	Scan z (0)(4)
(2)	E->F*	0	Comp$_1$	Zkompletováno z (0)(3)
(3)	S->E*	0	Comp$_1$	Zkompletováno z (0)(1)

(4)	E->E*QF	0	Act_1	Zkompletováno z (0)(2)
(5)	Q->*+	1	$Pred_1$	Predikce z (1)(4)
(6)	Q->*-	1	$Pred_1$	Predikce z (1)(4)
S(2): a-*a+a				
(1)	Q->-*	1	$Comp_2$	Scan z (1)(6)
(2)	E->EQ*F	1	Act_2	Zkompletováno z (1)(4)
(3)	F->*a	2	$Pred_2$	Predikce z (2)(2)
S(3): a-a*+a				
(1)	F->a*	2	$Comp_3$	Scan z (2)(3)
(2)	E->EQF*	1	$Comp_3$	Zkompletováno z (2)(2)
(3)	S->E*	0	$Comp_3$	Zkompletováno z (1)(3)
(4)	E->E*QF	0	Act_3	Zkompletováno z (0)(2)
(5)	Q->*+	3	$Pred_3$	Predikce z (3)(4)
(6)	Q->*-	3	$Pred_3$	Predikce z (3)(4)
S(4): a-a+*a				
(1)	Q->+*	3	$Comp_4$	Scan z (3)(5)
(2)	E->EQ*F	0	Act_4	Zkompletováno z (3)(4)
(3)	F->*a	4	$Pred_4$	Predikce z (4)(2)
S(5): a-a+a*				
(1)	F->a*	4	$Comp_5$	Scan z (4)(3)
(2)	E->EQF*	0	$Comp_5$	Zkompletováno z (4)(2)
(3)	S->E*	0	$Comp_5$	Zkompletováno z (0)(2)
(4)	E->E*QF	1	Act_5	Zkompletováno z (0)(3)

Tab. 4 Zpracování vstupního řetězce „a-a+a“ gramatikou A

Po zpracování posledního symbolu Scanerem ještě spouštíme Completer ke zpracování finálních redukcí, spouštět Prediktor již je zcela zbytečné, protože není co predikovat. Poznamenejme, že zpracování řetězce začalo voláním Prediktoru na počátečním $Active_0$ setu a dále pro každý symbol vstupního řetězce se provede jedna operace predikce / scan / completace. Protože poslední complete set obsahuje položku *S->E*,0,* je nalezena minimálně jedna úspěšná derivace vstupního řetězce z počátečního symbolu.

5.5.3 Konstrukce derivačního stromu vstupního řetězce

Z výše popsaného algoritmu přímo nejsme schopni získat derivační strom daného vstupního řetězce. Jay Earley ve svém článku z roku 1970 [6] poskytnul metodu pro konstrukci derivačního stromu v průběhu analýzy pomocí rozšíření položky o uchovávání zpětných ukazatelů, které ukazují na položku, jež zapříčinila její vznik. Ovšem Tomita [16] dokázal, že tato metoda může u nejednoznačných gramatik vést k nesprávným výsledkům. Pro vlastní rozpoznávání deformovaných objektů nejsou derivační stromy potřebné, mají význam pouze při návrhu a ladění vlastního algoritmu.

5.5.4 Analýza nejednoznačné gramatiky

Vytváření seznamu Earlyho stavů (*itemsets*) pro zpracování nejednoznačné gramatiky se neliší od zpracování gramatiky jednoznačné. Některé položky budou vloženy vícekrát do stejného seznamu, ale tato situace se může vyskytnout i při zpracování jednoznačné gramatiky. Pro vytvoření korektního derivačního stromu vstupního řetězce je nutné použít jiný způsob, například pomocí aplikace Unger parseru [25]. Vzhledem k podstatě

nejednoznačné gramatiky těchto derivačních stromů může být více. Výčet všech možných derivačních stromů je v řadě aplikací nezbytný, ovšem v řešené úloze rozpoznávání deformovaných objektů derivační stromy nemají téměř žádný význam. Jejich podoba je ovlivněná úpravou algoritmu pro akumulaci vah pravidel deformační gramatiky. Ze své podstaty jich může být více než jeden, ale pro správnou klasifikaci je důležitý pouze výsledek analýzy, respektive jeho výsledná váha.

5.5.5 Příklad analýzy nejednoznačné gramatiky

Následující příklad v tab. 5 ukazuje analýzu nejednoznačné gramatiky s pravidly S → SS a S → x a vstupního řetězce „xxx“. Z důvodů rozsáhlosti výpisu analýzy se jedná pouze o umělý příklad.

S(0): *xxx			
(1)	S->*SS	0	Act_0
(2)	S->*x	0	$Pred_0$
S(1): x*xx			
(1)	S->x*	1	$Comp_1$
(2)	S->S*S	1	Act_1
(3)	S->*SS	2	$Pred_1$
(4)	S->*x	2	$Pred_1$
S(2): xx*x			
(1)	S->x*	2	$Comp_2$
(2)	S->SS*	1	$Comp_2$
(3)	S->S*S	2	Act_2
(4)	S->S*S	1	Act_2
(5)	S->*SS	3	$Pred_3$
(6)	S->*x	3	$Pred_3$
S(1): xxx*			
(1)	S->x*	3	$Comp_3$
(2)	S->SS*	2	$Comp_3$
(3)	S->SS*	1	$Comp_3$
(4)	S->S*S	3	Act_3
(5)	S->S*S	2	Act_3
(6)	S->S*S	1	Act_3

Tab. 5 Zpracování vstupního řetězce „xxx“ nejednoznačnou gramatikou

5.5.6 Zpracování e-pravidel

Podobně jako velké množství parserů, Earlyho parser nedokáže zvládnout e-pravidla bez speciálních opatření. Problém nastává během predikce položek tvaru *A->*..., p+1* jako následek výskytu **A* v položce v seznamu $active_p$ nebo $predicted_p$, parser může narazit na prázdnou predikci *A->*, p+1*. Toto znamená, že neterminál *A* byl zkompletován právě před symbolem *p+1* a tato kompletace by měla být přidána do seznamu $completed_p$, který doposud obsahoval položky *p* nejvýše. Toto znamená nutnost spustit Completer znovu. Tímto ale obtíže nekončí. Pokud Completer spustíme znovu, zachytí nově přidané položky, které jsou úrovně *p + 1*. Completer bude tedy konzultovat $itemset_p$, který je nekompletní, protože se stále přidávají položky do jeho části $active_p$ a $predicted_p$. Pokud se v seznamu najde výskyt **A,* přidá se místo toho kopie *A** a takovéto položky mohou vyžadovat další predikce (pokud za * následuje další neterminál), část z nich mohou být již zkompletované

položky patřící do $completed_p$ a znamenající opět více práce pro Completer. Další položky mohou mít úroveň nižší než *p*, což přináší zpět vzdálenější položky, které mohou či nemusejí být zkompletovány… Nejjednodušší a obvykle zcela postačující cesta, jak tyto obtíže vyřešit, je nechat běžet Completer a Prediktor, dokud je co přidávat. Jediná nevýhoda tohoto řešení je zvýšení časové složitosti analýzy.

Další přístup byl navržen Earleym [6], [5], který navrhl, aby se v Completeru tečka posouvala přes neterminální symbol a následně došlo k vyhledání, zda se do S_i přidají nějaké položky. Z důvodu efektivnosti je nutné kolekci sledovaných neterminálů ukládat do datové struktury s rychlým přístupem. Tento způsob je relativně komplikovaný a nepříliš efektivní, podrobněji v [11].

Patrně nejlepší způsob zahrnuje jednoduchou modifikaci Prediktoru založenou na myšlence nulovatelnosti. Neterminál A je nulovatelný, pokud $A \rightarrow e$ (z *A* lze derivovat *e*), terminální symboly pochopitelně nulovatelné nejsou. Prediktor tedy může být popsán následovně:

- Pro každý stav v S(k) tvaru (X → α • Y β, j) (kde j je původní úroveň stavu), přidej (Y → • γ, k) do S(k) pro každé pravidlo s Y na levé straně. Pokud je Y nulovatelné, též přidej (X → α Y • β, j) do S(k).

Tedy tečka se přesune přes neterminál, pokud neterminál může derivovat prázdný symbol a tímto může „zmizet“, podrobněji v [11]. Seznam nulovatelných neterminálů je zvláště u deformačních gramatik velice jednoduchý, neboť se jedná o všechny neterminály.

Každý z popisovaných způsobů zajištění zpracování e-pravidel vede ke zvýšení časové a paměťové složitosti algoritmu. Proto pokud je možné, je výhodné se jejich zpracování vyhnout. U deformačních gramatik je zvláště výhodný způsob založený na myšlence nulovatelnosti. E-pravidla dokonce ani nemusí být v deformační gramatice vytvořena, stačí pouze, pokud jsou e-pravidla požadována, během operace Prediktoru přidávat stavy dle výše uvedeného předpisu.

5.5.7 Předpověď predikce

Pokud se více zaměříme na činnost Prediktoru, zjistíme, že některé předpovězené položky jsou bez dalšího užitku. Pokud by se nám podařilo snížit počet těchto zbytečných předpovědí, dosáhli bychom zvýšení časové i paměťové efektivity analýzy. Jedním ze způsobů, jak efektivitu zlepšit, je vzít v úvahu následující symbol vstupního řetězce. Například u gramatiky A z kapitoly 5.5.2, pokud další symbol je „-“, je zcela zbytečné předpovídat položku $Q \rightarrow *+,2$. Prediktor může být jednoduše modifikován k postižení těchto jednoduchých případů, je též možné vytvořit Prediktor, který nepředpoví nic na první pohled chybného, všechny jeho předpovězené položky budou v dalším setu kompletovány nebo aktivní. Ovšem predikce může selhat na následujícím symbolu. Informace potřebné k vytvoření takto perfektního Prediktoru mohou být získány spočítáním FIRST setu všech neterminálů [24, strana 27], [4] v gramatice. Použití FIRST setu je velice jednoduché: mějme jako vstupní řetězec pouze symbol „*q*“, Prediktor opět začíná od počáteční položky, ale protože ví, že symbol *q* není v FIRST(A), nepředpoví $S \rightarrow *A,0$. Položkám jako $A \rightarrow *C,1$ dokonce není potřeba se vyhýbat, jejich vytvoření nebude nikdy zvažováno. Pouze B-linie bude předpovězena.

S'-> S	
S -> A | AB | B	FIRST(S) = {ε, p, q}
A -> C	FIRST(A) = {ε, p}
B -> D	FIRST(B) = {q}
C -> p | ε	FIRST(C) = {ε, p}
D -> q	FIRST(D) = {q}

Tab. 6 Příklad umělé gramatiky a jejího FIRST setu

Zpracování e-pravidel je nyní jednodušší, protože známe pro každý neterminál, zda může produkovat *e* (podle toho, zda je v FIRST setu neterminálu). Bouckaert, Pirotte a Snelling analyzovali různé varianty Earleyho parseru pro dva rozdílné režimy předpovědi [14] a ukázali, že předpověď Prediktoru redukuje počet položek o 20-50% nebo dokonce více na některých praktických gramatikách.

5.5.8 Předpověď redukce

Druhým typem předpovědi je předpověď redukce, která redukuje počet kompletovaných položek – na rozdíl od předpovědi predikce, která redukuje počet predikovaných položek. Zaměřme se znovu na tab. 4, která obsahuje dvě zcela zbytečné kompletace v seznamu $completed_1$ položku $S \rightarrow E^*,0$ a $S \rightarrow E^*,0$ v seznamu $completed_3$. Redundance těchto položek vychází z faktu, že mají smysl pouze na konci vstupního řetězce. Toto se může zdát pouze jako velice specifický případ, kterým se nemá cenu zabývat, ale tento aspekt lze brát v mnohem širším významu.

Pokud zavedeme explicitní symbol pro konec vstupního řetězce, můžeme rozhodnout, že dané položky jsou zbytečné, protože jsou následovány symbolem, který není v množině symbolů položky následující za ní při kompletaci. Problém je udržovat, společně s položkou, onu množinu symbolů, které mohou následovat za položkou – množina symbolů pro předpověď redukce. Pokud se položka jeví být zkompletovaná, ale následující symbol není v této množině, je položka vyřazena. Pravidla pro konstrukci množiny symbolů, které mohou následovat za položkou, jsou zřejmá, ale na rozdíl od předpovědi predikce tato množina nemůže být zkonstruována dopředu, musí být konstruována a aktualizována během analýzy. Omezená a podstatně méně efektivní množina může být zkonstruována staticky, s využitím FOLLOW setu [24, strana 27], [4]. Efektivita této předpovědi se nedá jednoduše stanovit. Earley doporučuje předpověď redukce, ale nebere v úvahu obtížnost konstrukce a udržení množin předpovědi.

Bouckaert, Pirotte and Snelling [14] definitivně zavrhují předpověď redukce na základě toho, že může jednoduše zdvojnásobit počet položek, které je nutno uchovávat, ale počítají například $E \rightarrow *F\ [+\text{-}],1$ jako dvě položky. Závěrem lze říci, že zisk z předpovědi redukce není velký a její implementační náklady projevující se ve zvýšení časové a paměťové složitosti algoritmu, jež je významná, nevyváží úsporu vyplývající z redukce počtu kompletovaných položek. Například velmi dobře navržený Earley/CYK parser od Graham, Harrison a Ruzzo [26] nezahrnuje předpověď redukce.

5.5.9 Využití předpovědi pro zefektivnění analýzy nejednoznačné gramatiky pro rozpoznávání deformovaných objektů

Předpověď predikce v popsané podobě využívající FIRST množiny je u deformačních gramatik nepoužitelná, FIRST množiny vlastně udávají, jaké terminály je možno derivovat z jakých neterminálů. U deformačních gramatik je možné (již z podstaty deformační gramatiky) derivovat z neterminálu jakýkoliv terminál.

Předpověď redukce v popsané podobě je opět u deformačních gramatik nepoužitelná, množina symbolů, které mohou následovat za položkou, by byla vždy stejná a obsahovala by opět všechny terminály.

Ovšem i u deformačních gramatik lze nalézt způsoby, jak zmenšit počet generovaných položek. Pokud se zaměříme na jeden symbol vstupního řetězce, který odpovídá jednomu popisnému primitivu, je zcela zřejmé, že symbol následující za ním nemůže odpovídat primitivu opačnému, tedy například primitivum popisující úsečku vpravo nebude následováno primitivem popisujícím úsečku vlevo. Tento případ při popisu reálných objektů nemůže při vhodné volbě měřítka primitiv nastat a efektivitu analýzy tedy zvýšíme, pokud vyřadíme všechny položky vedoucí k danému opačnému primitivu na dalším místě. Nové položky do seznamu položek přináší operace predikce, tedy toto „filtrování“ bude umístěno v operaci predikce, kterou tedy nyní můžeme definovat následovně:

- Predikce: Pro každý stav v SListP(k) tvaru $(X \rightarrow \alpha \bullet Y\ \beta, j)$ (kde j je původní úroveň stavu), přidej $(Y \rightarrow \bullet\ \gamma, k)$ do *SListP(k)* pro každé pravidlo $Y \rightarrow \alpha$, kde α neobsahuje *NSign*, kde *NSign* je „opakem“ primitivy, kterou vyjadřuje aktuální vstupní symbol.

Operace S*can* pro *n-tý* symbol vstupního řetězce prochází seznam *SListP(n-1)* a hledá vhodné pravidlo tvaru $(X \rightarrow \alpha \bullet a\ \beta, j)$, kde a je onen *n-tý* symbol vstupního řetězce, tedy terminál. V deformační gramatice je pravidel vyhovujících tomuto zápisu více, liší se terminálem, pro který jsou. V *n-té* úrovni seznamu se použijí pouze ta, jejichž část za znakem Earlyho tečkové notace začíná *n+1* symbolem vstupního řetězce. Je tedy zbytečné v operaci predikce generovat položky, které začínají jiným terminálním symbolem, než je následující symbol ve vstupním řetězci. Finální definice operace predikce je tedy následující:

- Predikce: Pro každý stav v *SListP(k)* tvaru $(X \rightarrow \alpha \bullet Y\ \beta, j)$ (kde j je původní úroveň stavu), přidej $(Y \rightarrow \bullet\ \gamma, k)$ do *SListP(k)* pro každé pravidlo $Y \rightarrow \alpha$, kde α neobsahuje *NSign*, a kde γ je neterminál nebo terminál = *NextSign*; *NSign* je „opakem“ primitivy, kterou vyjadřuje aktuální vstupní symbol, *NextSign* je *k+1* symbol vstupního řetězce; $k \in 0..n-1$, kde n je délka vstupního řetězce

5.5.10 Optimalizace gramatiky pro konkrétní vstupní slovo

Earlyho parser ve své operaci predikce prochází přepisovací systém gramatiky a vyhledává pravidla určitých vlastností (viz popis algoritmu) a podle nich do seznamu stavů vkládá položky, které se nemusejí vždy využít. Při velkém počtu pravidel (typicky deformační gramatiky) významně narůstá režie při neustálém vyhledávání vhodné položky během provádění jednotlivých operací Earlyho parseru. Pokud zmenšíme počet pravidel gramatiky, zrychlí se prohledávání přepisovacího systému v operaci predikce a též se může

zmenšit počet zbytečně vložených položek, což opět vede ke zrychlení prohledávání seznamu stavů, ústících v celkové zrychlení algoritmu.

Jedno konkrétní vstupní slovo nemusí obsahovat všechny terminální symboly gramatiky, a tedy se při jeho analýze nemusí využívat celý přepisovací systém gramatiky. Pokud tato pravidla nepotřebná pro analýzu daného vstupního slova z gramatiky odstraníme, dosáhneme z výše uvedených důvodů větší efektivity analýzy. Algoritmus této optimalizace je naznačen na obr. 14.

```
function TParser.OptimizeGrammar(aGrammar: TGrammar; aInputString: string): TGRammar;
begin
  result := CopyGrammar(aGrammar);
  TerminalsToDelete := TList.Create;
  for i := 0 to aGrammar.Terminals.Count - 1 do begin
    T := aGrammar.Terminals.Items[i] as string;
    if not aInputString.Contains(T) then begin
      result.Terminals.Delete(result.Terminals.IndexOf(T));
      TerminalsToDelete.Add(T);
    end;
  end;
  for i := 0 to TerminalsToDelete.Count - 1 do begin
    T := TerminalsToDelete.Items[i] as string;
    for j := 0 to aGrammar.Rules.Count - 1 do begin
      TempR := aGrammar.Rules.Items[j] as TRule;
      if TempR.RHS.Contains(T) then result.Rules.Delete(result.fRules.IndexOf(TempR));
      //pokud jeho RHS obsahuje terminál nevyskytující se ve vstupním řetězci, toto
      //pravidlo vymažu z výsledné gramatiky....
    end;
  end;
  TerminalsToDelete.Free;
end;
```

Obr. 14 Algoritmus pro optimalizaci gramatiky

5.6 LR PARSERY

LR parser je parser pro bezkontextové gramatiky, který čte vstupní řetězec zleva doprava a produkuje nejpravější derivaci. Termín LR(k) parser je též často používán, *k* označuje počet *look ahead* symbolů, které se během parsovacího procesu využívají. Obvykle je $k = 1$ a je často vynecháváno. Bezkontextová gramatika je nazývána LR(k) gramatikou, pokud pro ni existuje LR(k) parser. LR parsery provádí *bottom-up* analýzu, protože se pokouší dedukovat počáteční symbol ze vstupního řetězce.

Množství programovacích jazyků lze popsat LR(1) gramatikou nebo gramatikou k ní blízkou, a proto jsou LR parsery často použity pro syntaktickou analýzu zdrojového kódu.

LR parsery mají řadu výhod:

- Množství programovacích jazyků může být analyzováno nějakou variantou LR parseru. Jedna důležitá výjimka je C++.
- LR parsery mohou být implementovány velmi efektivně.

- Ze všech parserů, které pracují se vstupem zleva doprava, LR parsery detekují syntaktické chyby (tedy že vstupní řetězec neodpovídá gramatice) tak brzo, jak je to jen možné.

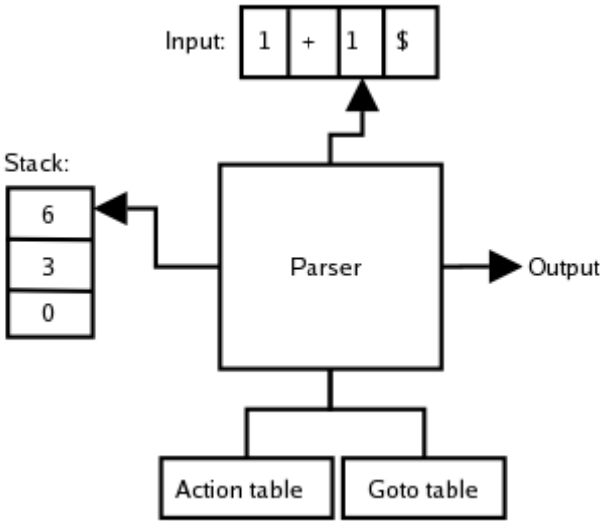
LR parsery je obtížné generovat „ručně“, obvykle jsou generovány nějakým parser generátorem. V závislosti na způsobu generování tabulek parseru (*action table, goto table*) jsou tyto parsery nazývány Simple LR parser (SLR) [4], Look-ahead LR parser (LALR) [2], [4] a Canonical LR [4] parser. Typ parseru určuje množinu gramatik, kterou je schopen zpracovat. Oblíbený Yacc produkuje LALR parsery.

5.6.1 Architektura LR parseru

LR parsery lze implementovat různými způsoby popsanými například v [4]. Obvykle se ale popisují a implementují jako tabulkově založené algoritmy využívající zásobník. Jeho schematické znázornění je na obr. 15.

Parser se skládá z:

- Vstupního bufferu *Input*.
- Zásobníku *Stack*, ve kterém je uložen seznam zpracovávaných stavů.
- *Goto table*, která popisuje přesuny do nových stavů.
- *Action table*, která udává pravidlo gramatiky, dané aktuálním stavem a aktuálním terminálem ve vstupním řetězci.



Obr. 15 Schematické znázornění architektury LR parseru

5.6.2 Algoritmus LR parseru:

1. Zásobník je inicializován s [0]. Aktuální stav je vždy stav na vrcholu zásobníku.
2. Podle aktuálního stavu a aktuálního terminálního symbolu vstupního řetězce je v action table vyhledána akce. Jsou zde čtyři možnosti:
   - *shift* (posun) s*n*:
     - aktuální terminál je odstraněn ze vstupního řetězce

- stav *n* je vložen do zásobníku a stává se současným stavem
- reduce (redukce) r*m*:
  - číslo *m* je zapsáno do výstupního řetězce
  - pro každý symbol na pravé straně pravidla *m* je jeden stav odstraněn ze zásobníku
  - aktuálním stavem na vrcholu zásobníku a levou stranou pravidla *m* je vyhledán nový stav v *goto table* a je vložen na vrchol zásobníku a tímto se současně stává stavem aktuálním
- accept: vstupní řetězec je akceptován
- žádná akce: je hlášena syntaktická chyba

3. Krok 2 je opakován, dokud není vstupní řetězec akceptován, nebo není hlášena vstupní chyba.

5.6.3 Konkrétní příklad

K vysvětlení, jak algoritmus pracuje, použijeme následující gramatiku B (obr. 16) s počátečním symbolem *E* a uvažujme vstupní řetězec „*1+1*“

(1) E → E * B
(2) E → E + B
(3) E → B
(4) B → 0
(5) B → 1

Obr. 16 Gramatika B

Dvě LR(0) tabulky parseru pro tuto gramatiku jsou znázorněny v tab. 7.

	Action table						Goto table	
Stav	*	+	0	1	$		E	B
0			s1	s2			3	4
1	r4	r4	r4	r4	r4			
2	r5	r5	r5	r5	r5			
3	s5	s6			acc			
4	r3	r3	r3	r3	r3			
5			s1	s2				7
6			s1	s2				8
7	r1	r1	r1	r1	r1			
8	r2	r2	r2	r2	r2			

Tab. 7 Action a Goto tabulky pro parser pro gramatiku B

Action table je indexována stavem parseru a terminálem (včetně speciálního symbolu „$“, označujícího konec vstupního řetězce) a obsahuje tři typy akcí:

- Shift (posun), který je zapsán jako „s*n“* a indikuje, že další stav je *n.*
- Reduce (redukce), která je zapsána jako „r*m“* a indikuje, že následuje redukce pravidla *m.*

- Accept (přijetí řetězce), které je zapsáno jako „acc“ a indikuje, že parser akceptuje vstupní řetězec.

Goto table je indexována stavem parseru a neterminálem a indikuje, jaký je další stav parseru v případě rozpoznání určitého neterminálu.

Stav	Vstupní řetězec	Výstupní řetězec	Zásobník	Další akce
0	1+1$		[0]	Shift 2
2	+1$		[0,2]	Reduce 5
4	+1$	5	[0,4]	Reduce 3
3	+1$	5,3	[0,3]	Shift 6
6	1$	5,3	[0,3,6]	Shift 2
2	$	5,3	[0,3,6,2]	Reduce 5
8	$	5,3,5	[0,3,6,8]	Reduce 2
3	$	5,3,5,2	[0,3]	Accept

Tab. 8 Rozpoznávání řetězce „*1+1*“

Celý proces rozpoznávání vstupního řetězce je znázorněn v tab. 8. Parser začíná pouze s počátečním stavem (0) vloženým do zásobníku.

[0]

První symbol ze vstupního řetězce, který parser vidí je „*1*“. Za účelem nalezení správné následující akce (shift, reduce, accept, error) je action table indexována s aktuálním stavem (ten, který je danou chvíli na vrcholu zásobníku), což je momentálně „0“, a aktuálním symbolem, jenž je momentálně „1“. Pro tyto hodnoty action table specifikuje operaci *shift2*, tedy stav 2 je vložen do zásobníku. Výsledný zásobník vypadá:

[0 2]

Ve stavu 2 říká action table bez ohledu na terminál, který vidíme ve vstupním řetězci, že bychom měli provést redukci s pravidlem gramatiky 5. Pokud je tabulka správná, znamená to, že parser právě rozpoznal pravou stranu pravidla číslo 5. Zapíšeme 5 do výstupního řetězce, vyjmeme jeden stav ze zásobníku (pravá strana pravidla 5 má pouze jeden symbol) a vložíme do zásobníku stav z buňky v goto table pro stav 0 neterminál B, tedy stav 4. Výsledný zásobník vypadá:

[0 4]

Ve stavu 4 action table říká, že bychom měli provést redukci s pravidlem 3. Tedy zapíšeme 3 do výstupního řetězce, vyjmeme jeden stav ze zásobníku a v goto table vyhledáme nový stav pro stav 0 a neterminál E, což je stav 3, který vložíme do zásobníku. Výsledný zásobník vypadá:

[0 3]

Další terminální symbol, který parser vidí, je „+“ a podle action table bychom měli provést operaci *shift6:*

[0 3 6]

Další terminál je nyní „1“, což znamená, že provádíme operaci *shift2*:

[0 3 6 2]

Stejně jako předchozí „1“ je i tato redukována na neterminál B, což ústí v následující zásobník:

[0 3 6 8]

Ve stavu 8 vždy provádíme redukci s pravidlem 2, jehož pravá strana má 3 symboly a tedy ze zásobníku vyjímáme 3 stavy. Pro stav 0 a neterminál E v goto table nalezneme stav 3.

[0 3]

Konečně můžeme přečíst „$“ ze vstupního řetězce, což podle action table znamená, že parser akceptuje vstupní řetězec. Čísla pravidel, která byla zapsána do výstupního řetězce [5, 3, 5, 2], popisují nejpravější derivací řetězce „*1+1*“.

5.6.4 Konstrukce tabulek parseru

Největším problémem při vytváření parseru je konstrukce jeho tabulek, která je obecně velmi náročná. Tabulky se obvykle vytváří před analýzou nějakého vstupního řetězce, často za pomoci takzvaného parser generátoru. Díky takto předem vytvořeným tabulkám je vlastní analýza velice rychlá. Konstrukce tabulek LR(0) je popsána například v [1], další typy jsou naznačeny například v [4]. Tedy existuje jeden algoritmus pro syntaktickou analýzu, ale více algoritmů pro konstrukce tabulek parseru. Základní algoritmy pro konstrukci tabulek:

- Simple LR (SLR) – nejslabší, ale jednoduchý a vytváří málo stavů.
- Canonical LR – silnější, ale vytváří poměrně hodně stavů.
- Lookahead LR – (LALR) – nejlepší, protože je nejsilnější a přitom vytváří stejný počet stavů jako SLR.

5.6.5 Konflikty v tabulkách

Tabulky parseru musí být jednoznačné, tedy jedna buňka tabulky nemůže obsahovat dvě akce. V případě, že obsahuje, hovoříme o takzvaném konfliktu v tabulce. Obvykle jsou způsobeny nějakou nejednoznačností v gramatice. Mohou být:

- *Shift/reduce conflict* – obvykle indikuje nejednoznačnou konstrukci v gramatice (typický příklad – *dangling else* v Pascalu [22]), můžeme eliminovat modifikací gramatiky nebo upřednostněním shift operace. Konflikt je ilustrován na obr. 17.
- *Reduce/reduce conflict* – opět indikuje nejednoznačnou konstrukci v gramatice, často nemá žádné jednoduché řešení, někdy je možné vytvořit jiný blízký jazyk jako podmnožinu požadovaného jazyka, který bude bez konfliktu. Konflikt je ilustrován na obr. 17.

(1) E → 1 E
(2) E → 1

Obr. 17 Příklad non-LR(0) gramatiky s shift-reduce konfliktem

(1) E → A 1
(2) E → B 2
(3) A → 1
(4) B → 1

Obr. 18 Příklad non-LR(0) gramatiky s reduce-reduce konfliktem

Oba příklady konfliktů na obr. 17 a obr. 18 by šlo vyřešit použitím SLR parseru.

5.6.6 Použití LR parserů pro analýzu deformačních gramatik

Použití parserů z rodiny LR pro analýzu deformační gramatiky je nemožné, protože nejednoznačná deformační gramatika produkuje v tabulkách parseru neřešitelné konflikty, které ani jediná další informace, kterou máme, váhy jednotlivých pravidel nemohou odstranit. Ovšem existují další parsery, které LR parser používají nebo z něj vycházejí a dokážou analyzovat nejednoznačnou deformační gramatiku. Proto byl zde věnován prostor jeho základnímu popisu.

5.7 TOMITA PARSER

Z předchozí kapitoly je zřejmé, že existují gramatiky, které LR parsery nejsou schopné analyzovat. Ovšem Tomita [16] popsal dobře fungující a velmi efektivní metodu, pomocí které je možné analyzovat i takové gramatiky.

Tomitovu metodu lze jednoduše popsat jako prohledávání do šířky [4] v místech analýzy, které LR parsery (LR(1), LALR(1), SLR(1), LR(0)), precedenční [4], či ještě jednodušší nemohou zpracovat kvůli konfliktům, zatímco současně jsou uchovávány částečné derivační stromy. Přesněji, pokaždé když se v průběhu analýzy na vrcholu zásobníku objeví neadekvátní stav, jsou podniknuty následující kroky:

1. Pro každou možnou redukci ve stavu je udělána kopie zásobníku a daná redukce je provedena. Toto odstraní pravý konec zásobníku a nahradí jej neterminálem, který použijeme k vyhledání nového stavu parseru, jenž vložíme na vrchol zásobníku. Pokud tento stav opět umožňuje redukci, je tento krok opakován, dokud nejsou zpracovány všechny možné redukce, což ústí v příslušný počet kopií zásobníku.
2. Zásobníky, které mají na vrcholu stav neumožňující operaci shift na následujícím vstupním symbolu, jsou zahozeny (protože mají původ v chybných odhadech). Kopie následujícího vstupního symbolu jsou vloženy do zbývajících zásobníků.

Pokud parser používá look-ahead, je ho potřeba vzít v úvahu během provádění kroku 1. Ignorování look-ahead informace nevede k chybným výsledkům, pouze snižuje efektivitu algoritmu – bude zkopírováno a následně vyřazeno více zásobníků. Dále pokud má gramatika smyčky (pravidla ve tvaru $A \rightarrow B$, $B \rightarrow A$), může krok jedna alternovat mezi A a B a nikdy neskončit. Jsou zde dvě možná řešení:

- Při vytváření zásobníku zkontrolovat, zda již neexistuje, a případně ignorovat.
- Předem zkontrolovat gramatiku, zda neobsahuje smyčky.

Pokud jsou v kroku dva zahozeny všechny zásobníky, je vstup chybný – v onom konkrétním místě. Jednoduché kopírování zásobníků může způsobit přílišné rozšíření zásobníků se sklonem k duplikování mnoha informací, což není žádoucí. Proto se v praktické verzi parseru používají metody optimalizace kopírování zásobníků:

- kombinace stejných stavů (*combining equal states*)
- kombinace stejných začátků zásobníků (*combining equal stack prefixes*)

Jejich podrobnější popis lze nalézt například v [4]. Metoda bude pracovat s jakoukoliv *bottom-up* tabulkou nebo i dokonce bez jakékoliv tabulky. Čím slabší tabulku použijeme, tím více nedeterminismu bude potřeba zpracovat pomocí prohledávání do šířky a při chybějící tabulce metoda degeneruje na prosté prohledávání do šířky. Časové požadavky [4] této metody jsou v principu exponenciálně úměrné délce řetězce, prakticky jsou ale mnohem skromnější, obecně lineární nebo o něco více než lineární a téměř vždy menší než požadavky Earlyho parseru nebo CYK parseru s výjimkou velmi nejednoznačných gramatik. Tomita parser je zvláště efektivní, pokud je v tabulkách pouze několik málo konfliktů.

Tomita parser je vhodný, pokud chceme pouze rozpoznat vstupní text. Avšak pokud chceme vykonat nějaké sémantické akce, narazíme na problém současného běhu více parserů. Možné řešení je pro každou kopii LR parseru konstruovat derivační strom. Na konci analýzy vstupního symbolu můžeme tyto stromy zpracovat k vykonání požadovaných sémantických akcí. Ovšem výpočetní náročnost vytváření a následného zpracování těchto derivačních stromů téměř neguje výhody Tomita parseru.

5.7.1 Modifikace Tomita parseru pro rozpoznávání deformovaných objektů

Z předchozí kapitoly je zřejmé, že Tomita parser je možno použít pro analýzu nejednoznačné deformační gramatiky, tedy umožňuje využít tabulky parseru rodiny LR, které se vyznačují svojí efektivitou. Nejdůležitější modifikací parseru je nutnost akumulovat váhy deformačních pravidel k získání výsledné vzdálenosti analyzovaného řetězce od obrazu dané třídy objektu. Tohoto dosáhneme založením proměnné, jež na začátku analýzy nastavíme na hodnotu 0 a při každé redukci přičteme váhu redukovaného pravidla. Při vytváření kopie zásobníku (zpracování konfliktu v tabulce parseru) se zkopíruje i váhovací proměnná a výsledek analýzy je úspěšná / úspěšné derivace vstupního řetězce s nejmenší hodnotou váhovací proměnné. Připomeňme, že při analýze deformační gramatiky nás zajímá pouze úspěch / neúspěch a hodnota váhovací proměnné.

Je možné (nevede k chybě) vytvářet kopie zásobníků pro všechny možné redukce. Ale lze využít váhy jednotlivých pravidel možných v daném okamžiku pro redukci a vybrat pouze pravidlo / pravidla s nejmenší vahou.

Příklad analýzy zde není uveden z důvodů přílišné rozsáhlosti. Popsaný algoritmus je mírně efektivnější a „šikovnější“ než Earlyho parser, jeho míru efektivity lze přesně stanovit jen obtížně. Oproti Earlymu parseru například nemá problém se zpracováním e-pravidel (nutno použít tabulky parseru LR(1) nebo lepší). Do testovacího prostředí tento algoritmus nebyl zahrnut z důvodů potřeby vytvoření tabulek parseru k dané gramatice, což je značně náročné a použití nějakého parser generátoru je díky komplikaci v podobě vah pravidel deformační gramatiky téměř vyloučeno.

5.8 MODIFIKOVANÝ HYBRIDNÍ LRE(K) ALGORITMUS

Další způsob, jak obejít konflikty v parsovacích tabulkách LR(k) parseru pro obecnou bezkontextovou gramatiku, spočívá ve využití podobnosti LR(k) a Earlyho parseru. Tato varianta Earlyho parseru [21] podobně jako Tomita parser využívá LR parsovací tabulky pro zajištění efektivnosti, zatímco si ponechává výhodu dovolující jednoduše přiřadit ke gramatice i nějaké sémantické akce. LR tabulky jsou vlastně předem vypočítané veškeré

operace, které Earlyho parser vykonává během své činnosti. Hybridní LRE(k) parser využívá informace z LR tabulek a vyhýbá se tedy výpočtu těchto informaci za běhu parseru. Jméno parseru LRE(k) odráží skutečnost, že parser je vytvořen jako kombinace LR(k) parseru a Earlyho parseru.

Klasický Earlyho parser nabízí relativně snadnou implementaci. Ovšem při analýze jeho činnosti je patrno, že parser tráví většinu času během své funkce vytvářením nových položek během operace kompletace. Spousta předpovězených položek nemusí být využita v průběhu analýzy vstupního řetězce. Dále některé Earlyho položky mohou být seskupeny způsobem, který využívá předvypočtené vlastnosti gramatiky. Položky se seskupují stejným způsobem jako stavy deterministického (a pravděpodobně neadekvátního) LR(k) parseru.

V následujícím popisu je použito $X_1, X_2 \ldots X_n$ k reprezentaci symbolů vstupního řetězce. Pokud jsou lookahead množiny správně definovány, předpokládáme, že vstupní řetězec je ukončen k hraničními znaky „@“. Tedy X_{n+i} = „@“, pro $1 < i < k$. Činnost hybridního algoritmu je založena na činnosti Earlyho parseru a může být popsána porovnáním jeho činnosti s činností klasického Earlyho parseru. Klasický Earlyho parser používá položky tvaru $[A \rightarrow \alpha \bullet \beta, t_1 \ldots t_k, p]$, kde $A \rightarrow \alpha \bullet \beta$ je pravidlo se znakem Earlyho tečkové notace, $t_1 \ldots t_k$ je lookahead pro danou položku a p je odkaz zpět na úroveň, kde rozpoznávání pravidla $A \rightarrow \alpha \beta$ začalo. Algoritmus využívá výhody, že první dvě komponenty Earlyho položky reprezentují položku v jednom nebo více stavech LR(k) parseru. Proto jsou stavy LRE parseru implementovány ve formě stavu LR(k) parseru, což přináší následující výhody:

- Můžeme využít LR(k) parsovací tabulky k určení akcí Earlyho parseru.
- Lookahead řetězce nejsou počítány dynamicky.
- Nová reprezentace algoritmu může být implementována způsobem, který spotřebovává méně paměti (oproti Earlyho parseru až poloviční úspora paměti).

Tento hybridní parser stále dokáže zpracovat obecné bezkontextové gramatiky a je 10 až 15krát rychlejší než Earlyho parser.

5.8.1 Popis LRE(k) parseru

Stav LRE parseru se nazývá Earlyho stav a je zapsán jako E_m. Stav E_m je dosažen po rozpoznání znaků řetězce $X_1, X_2, \ldots, X_{m-1}$. Stav E_m je reprezentován jako množina dvojic $\{<I_1, B_1>, <I_2, B_2>, \ldots\}$, kde každé $I_i \in S$ je číslo nějakého stavu LR(k) parseru a B_i je kolekce zpětných ukazatelů na Earlyho stavy. Každé B_i může být implementováno jako pole seznamů čísel LRE stavů, kde elementy pole odpovídají s položkami v LR(k) stavu I_i. Tedy je možno B_i reprezentovat seznamem seznamu čísel $[[b_{i11}, b_{i12}, b_{i13}, \ldots], [b_{i21}, b_{i22}, \ldots], \ldots, [b_{in1}, b_{in2}, \ldots]]$, kde každé číslo b_{ixy} je celé číslo z rozsahu 0 až k včetně a LR(k) stav I má n položek. Například: LRE stav E_3 má následující reprezentaci:

{<17, [[1, 2], [3], [3]]>, <23, [[2]]>}

Toto znamená, že stav E3 reprezentuje směs položek z LR(k) stavů číslo 17 a 23. Stav 17 musí mít tři položky, například:

- $A \rightarrow A \bullet B\ C\ \alpha_1$

- $X \rightarrow a\ A \bullet D\ \alpha_2$
- $A \rightarrow \bullet\ b\ \alpha_3$

Kde α_1, α_2 a α_3 jsou lookahead řetězce. Podobně LR(k) stav 23 musí mít pouze jednu položku:

- $C \rightarrow a\ b \bullet b\ \beta_1$

Nyní je celý LRE stav reprezentován Earlyho stavem, který obsahuje přesně tyto položky:
{<A->A*BC, a_1, 1>, <A->A*BC, a_1, 2>, <X->aA*D, a_2, 3>, <A->*b, a_3, 3>, <C->ab*b,b_1,2>}

První dvojice v E_3 reprezentuje dvě kopie LR stavu 17, kde jedna kopie je asociovaná s ukazatelem zpět na stav 1 a druhá s ukazatelem zpět na stav 2. Podobně i pro ostatní položky v LR stavech 17 a 23.

LRE algoritmus je založen na Earlyho algoritmu, ale je modifikován pro práci s odlišnou reprezentací stavů. Má dvě hlavní operace *Scan* a *Recognizer*. Z daného LRE stavu E_S operace *Scan*(E_S, *X, t)* vytvoří nový LRE stav, který reprezentuje Earlyho položky se znakem Earlyho tečkové notace posunutým za znak *X* ve všech Earlyho položkách ze stavu E_S, kde je to možné. Její kód je naznačen na obr. 19. Operace *Recognizer*($X_1,\ldots,X_n,\ldots,X_{n+k}$) rozhodne, zda řetězec znaků $X_1 \ldots X_n$ je možné generovat gramatikou G. Poznamenejme, že každý ze symbolů X_{n+1}, X_{n+2}, ... , X_{n+k} je symbol *„$"*. Tyto extra symboly jsou nezbytné pro provedení finálních redukcí. Recognizer konstruuje sekvenci Earlyho stavů, ze kterých též může být vytvořen korektní derivační strom. Její kód je na obr. 20. Více o tomto algoritmu lze najit v [21].

```
function Scan(Es, X, t)
begin
  result := Æ;
  for origin := každá dvojice in Es do begin
    dest := Shift[origin.State,X];
    if dest ³ 0 then begin
      newTuple := < dest, emptyBackPtrArray >;
        for i := 1 to NumberOfItems[origin] do begin
        j := DestItemPosition[origin,i];
        if j >= 0 then newTuple.BackPtrs[j]:= origin.BackPtrs[i]
      end;
      for j := 1 to NumberOfItems[dest] do begin
        if newTuple.BackPtrs[j] = empty then newTuple.BackPtrs[j] := [t]
      end;
      result := MERGE1(result, newTuple)
    end;
  end;
return result;
end
```

Obr. 19 Algoritmus operace Scan

```
function Recognizer(x1 ... xn+k)
begin
  E0 := { < Iinitial, [[0]] > };
  E1 := Scan(E0, £, 1);
```

```
  for i = 1 to n do begin
    Ei+1 := Scan(Ei, xi, i+1);
    repeat
      for LS := každý element v Ei+1 do begin
       rs := ReduceItemList(LS.State, xi+1xi+2...xi+k);
       for i := každý element v rs do begin
         lhs := LeftHandSymbol[LS.State,i];
         for j := každý element v LS.BackPtrs[i] do
           Ei+1 := Merge(Ei+1,Scan(Ej,lhs,i+1));
       end
      end
    until Ei+1 se nezmění;
    if Ei+1 = 0 then return neúspěch;
  end
  if En+1 = { < Iaccept, [[0]] > } then return úspěch
                                   else return neúspěch;
end;
```

Obr. 20 Algoritmus operace Recognizer

5.8.2 Příklad činnosti LRE(k) parseru

Mějme následující nejednoznačnou gramatiku:

1. $E \rightarrow E + E$
2. $E \rightarrow n$

Augmentujeme ji pravidlem 0. $S \rightarrow @\ E\ @$. Pro jednoduchost volíme $k = 0$. Z této gramatiky můžeme vytvořit tabulky LR(0) parseru, viz tab. 9. Tabulka obsahuje konflikty, ve stavu 7 implicitně obsahuje dvě odlišné akce pro symbol „+“.

Stav	Číslo položky	Položka	Akce	
1	1	[$S \rightarrow \bullet\ @\ E\ @$]	@	Shift 2
2	1	[$S \rightarrow @\ \bullet\ E\ @$]	E	Shift 3
	2	[$E \rightarrow \bullet\ E + E$]	E	Shift 3
	3	[$E \rightarrow \bullet\ n$]	n	Shift 4
3	1	[$S \rightarrow @\ E \bullet @$]	@	Shift 6
	2	[$E \rightarrow E \bullet + E$]	+	Shift 5
4	1	[$E \rightarrow n \bullet$]	Jakýkoliv	Reduce 2
5	1	[$S \rightarrow @\ E\ @ \bullet$]	Jakýkoliv	Reduce 0
6	1	[$E \rightarrow E + \bullet E$]	E	Shift 7
	2	[$E \rightarrow \bullet\ E + E$]	E	Shift 7
	3	[$E \rightarrow \bullet\ n$]	n	Shift 4
7	1	[$E \rightarrow E + E \bullet$]	Jakýkoliv	Reduce 1
	2	[$E \rightarrow E \bullet + E$]	+	Shift 6

Tab. 9 Tabulka LR(0) parseru

Z této LR(0) tabulky lze odvodit tabulky potřebné pro činnost LRE parseru. Význam jednotlivých sloupců tabulek:

- State – číslo stavu LR(k) parseru.
- ItemNo – číslo položky LR(k) stavu.
- DestItemPosition[m, i] – udává shodu mezi položkami LR(k) stavech. Pokud položka *i* v LR(k) stavu číslo *m* je $A \rightarrow \alpha \bullet X \beta$, pak posun na symbol *X* povede k unikátnímu LR(k) stavu, který obsahuje položku $A \rightarrow \alpha X \bullet \beta$. Číslo držené v DestItemPosition[m, i] je číslo této položky v cílovém stavu. Pokud položka *i* ve stavu *m* nemá požadovanou formu (znak Earlyho násobení je na konci pravé strany), předpokládáme, že DestItemPosition[m, i] má hodnotu -1.
- LeftHandSymbol[m, i] – udává symbol, který se objevuje na levé straně *i-té* položky v LR(k) stavu *m.*
- NumberOfItems[i] – udává počet položek v LR(k) stavu.
- ReduceItemList[m, a] – je seznam pozic všech položek v LR(k) stavu *m,* kde znak Earlyho tečkové notace je na konci pravé strany a kde look-ahead řetězec pro tyto položky je *a.*
- Symbol – vstupní symbol řetězce.
- Shift[s, x] – udává Shift akci s LR(k) parseru, pokud platná Shift akce *s* není definována pro symbol *x*, Shift[s, x] má hodnotu -1.

State	ItemNo.	DestItemPosition	LeftHandSymbol
1	1	1	S
2	1	1	S
2	2	2	E
2	3	1	E
3	1	1	S
3	2	1	E
4	1	-1	E
5	1	-1	S
6	1	1	E
6	2	2	E
6	3	1	E
7	1	-1	E
7	2	1	E

State	NumberOfItems	ReduceItemList
1	1	[]
2	3	[]
3	2	[]
4	1	[1]
5	1	[]
6	3	[]
7	2	[1]

State	Symbol	Shift
1	@	2
2	E	3
2	n	4
3	+	6
3	@	5
6	E	7
6	n	4
7	+	6

Tab. 10 Tabulky LRE parseru použité v příkladu činnosti

Ve sloupcích DestItemPosition a LeftHandSymbol jsou uvedeny pouze platné elementy (k chybějícím elementům by ovšem nikdy nemělo být přistupováno). Podobně v sloupci Shift jsou uvedeny pouze platné elementy, přístup k jakémukoliv jinému elementu vrací hodnotu -1.

Činnost LRE(0) parseru si demonstrujeme na vstupním řetězci „*n*+*n*+*n*“. Operace Recognizer je inicializována s množinou E_0 s počátečním LRE stavem {<1, [0]>}. Tento stav reprezentuje položku 1 stavu 1 LR(0) parseru, tedy že budeme rozpoznávat pravou stranu pravidla S → $ E $**.**

Každý číslovaný krok v průběhu procesu rozpoznávání odpovídá zpracování jednoho vstupního symbolu a spočívá vlastně ve vytvoření příslušného LRE stavu. Podrobný popis vytvoření LRE stavu je pouze u několika prvních kroků.

1. Zpracování počátečního symbolu **$**

E_1 = {<2, [[0], [1], [1]]>} Recognizer volá Scan(E_0, @, 1), který vyhledá akci pro LR(0) stav 1 a vstup $. Tedy je vytvořen LRE stav <2, [[], [], []]> a jsou naplněny zpětné ukazatele. Ukazatel [0] byl zkopírován z původního stavu a dva ukazatele obsahující [1] odpovídají kompletovaným položkám.

2. Zpracování prvního symbolu n

E_2 = {<4, [[1]]>, <3, [[0], [1]]>} Recognizer volá Scan(E_1, n, 2). Element <4, [[1]]> je vytvořen kvůli LR(0) akci pro stav 2 a look-ahead symbol n. Druhá položka je vytvořena Recognizerem, protože položka 1 v LR(0) stavu 4 je redukovaná položka a tato redukce je spuštěna, neboť následující vstupní symbol je „+“, LHS symbol pro tuto položku je E a tedy Recognizer volá Scan(E_1, E, 1) k vytvoření těchto dvou položek.

3. Zpracování vstupního symbolu +

E3 = {<6, [[1], [3], [3]]>}

4. Zpracování vstupního symbolu n

E4 = {<4, [[3]] >, <7, [[1], [3]]>, <3, [[0], [1]]>}

5. Zpracování vstupního symbolu +

E5 = {<6, [[1,3], [5], [5]]> }

6. Zpracování vstupního symbolu n

E6 = {<4, [[5]]>, <7, [[1,3], [3,5]]>, <3, [[0], [1]]>}

7. Zpracování koncového symbolu $

E7 = {<5, [[0]]>}

5.8.3 Modifikace LRE(k) parseru pro rozpoznávání deformovaných objektů

Z předchozích kapitol je vidět, že hybridní LRE parser je schopen analyzovat obecnou bezkontextovou gramatiku a je tedy možné ho využít pro analýzu nejednoznačné deformační gramatiky. Nezbytnou modifikací parseru je nutnost akumulovat váhy deformačních pravidel k získání výsledné vzdálenosti analyzovaného řetězce od obrazu dané třídy objektu. Tohoto dosáhneme způsobem podobným jako u modifikovaného Tomita parseru. Založíme proměnnou, její hodnotu na začátku analýzy nastavíme na hodnotu 0 a při každé redukci přičteme váhu redukovaného pravidla. Připomeňme, že při analýze deformační gramatiky nás zajímá pouze úspěch / neúspěch a hodnota váhovací proměnné.

Příklad analýzy deformační gramatiky zde opět není uveden z důvodů přílišné rozsáhlosti. Popsaný algoritmus je efektivnější a „šikovnější“ než Earlyho parser, jeho míru efektivity lze přesně stanovit jen obtížně. Oproti Earlymu parseru například nemá žádný problém se zpracováním e-pravidel (nutno využít tabulku LR(1) nebo lepší). Do testovacího prostředí tento algoritmus nebyl zahrnut z důvodů potřeby vytvoření tabulek parseru k dané gramatice, což je značně náročné a použití nějakého parser generátoru je díky komplikaci v podobě vah pravidel deformační gramatiky téměř vyloučeno.

6 ZAJIŠTĚNÍ INVARIANCÍ U METOD SYNTAKTICKÉ ANALÝZY

Metody syntaktické analýzy jsou invariantní vůči translaci popisovaného objektu, další invariance již ale musíme zajistit dodatečně. Jedná se především o invarianci vůči rotaci a invarianci vůči volbě počátečního bodu popisu objektu.

Invariance vůči volbě počátečního bodu popisu zajistí rozpoznání objektu při jakékoliv volbě bodu hrany objektu jako počátečního bodu popisu. V úvahu připadají následující dva způsoby:

- rotace řetězce popisujícího objekt
- automatická volba bodu počátku popisu

Invariance vůči rotaci způsobí, že metoda zajistí rozpoznání předloženého objektu v jakémkoliv jeho natočení. Invariance vůči rotaci patří k nezbytným vlastnostem téměř každého algoritmu pro rozpoznávání objektů. Syntaktické metody nejsou standardně invariantní vůči rotaci, je tedy nutné tuto invarianci nějakým způsobem zajistit. V úvahu připadají následující dva způsoby:

- využití diferenciálních primitiv pro popis objektů
- rotace objektů

Způsoby (například rotace předkládaného objektu a rotace řetězce popisujícího rozpoznávaný objekt) běžně používané u klasických strukturálních metod jsou použitelné i při rozpoznávání deformovaných objektů pomocí deformačních gramatik, ale zvyšují časovou složitost analýzy, která je významně vyšší než u klasické syntaktické analýzy. Dochází též i ke komplikaci při vyhodnocování výsledků z důvodu velkého počtu výsledných hodnot. Patrně jediná nepoužitelná výjimka zde je normalizace předkládaných řetězců [35], která se z důvodu proměnné délky řetězce popisujícího deformovaný objekt nedá použít.

Je nutné zvolit způsob, který současně umožní jednoduše zajistit potřebné invariance a zároveň způsob, který co nejméně navýší časovou složitost při zachování jednoduchého vyhodnocování výsledků analýzy.

Dále někdy mohou být požadovány invariance vůči změně měřítka a invariance vůči stranovému převrácení.

6.1 ZAJIŠTĚNÍ INVARIANCE VŮČI VOLBĚ POČÁTEČNÍHO BODU POPISU

První způsob zajištění této invariance je pomocí rotace řetězce popisujícího objekt. Rotací řetězce je myšleno přesunutí posledního terminálního symbolu na začátek.

$$abcde \overset{1.rotace}{\rightarrow} eabcd \rightarrow \cdots$$

Jsou analyzovány všechny možnosti volby počátečního bodu a je vybrána ta nejlepší, což u deformačních gramatik je ta s nejmenší váhou. Toto řešení je jednoduché a spolehlivé. U klasických syntaktických metod je zcela postačující, ovšem při analýze deformační gramatiky, jejíž doba analýzy je znatelně vyšší, není nejvhodnější. Potřebný čas je úměrný n

násobku doby analýzy, kde *n* je délka analyzovaného řetězce, tedy roste exponenciálně s délkou analyzovaného řetězce.

Druhý způsob spočívá, že řetězce popisující objekty jsou k analýze předkládány vždy se stejně zvoleným bodem počátku popisu, například bod objektu, který má maximální x a minimální y souřadnice (tedy bod objektu, který je nejvíce „vpravo nahoře“). Největší výhodou tohoto způsobu je jeho rychlost, protože celková doba analýzy je blízká době analýzy bez zajištění této invariance. Je pouze vyšší o dobu „režie“ algoritmu volby počátečního bodu popisu. Algoritmus této metody zajištění invariance vůči volbě počátečního bodu popisu je naznačen na obr. 21.

```
function NormalizeBeginOfDescription(aPattern: string; aStep: integer): string;
   var Points: TList;          //seznam souřadnic bodů objektu
       i: integer;             //pomocná iterační proměnná
       TempPoint: TPoint;      //pomocná proměnná reprezentující souřadnice jednoho bodu
       MaxIndex: word;         //proměnná uchovávající index nejlepšího bodu
       MaxX, MaxY: integer;     //pomocné proměnné pro porovnávání souřadnic při hledání
nejlepšího bodu
   begin
     //inicializace a vytvoření proměnných
     result := aPattern;       //inicializace výsledku
     Points := TList.Create;   //vytvoření seznamu bodů
     TempPoint.X := 30000;     //inicializace počátku „kreslení“
     TempPoint.Y := 30000;
     //nakreslí objekt do seznamu - jde o výpočet souřadnic všech jeho bodů
     //začnu od počátečního bodu a pokračuji dle primitiv
     //body uložím do seznamu Points
     MaxY := 60000;                              //taková souřadnice nebude
     for i := 0 to aPattern.Length-1 do begin    //procházím všechny body
       TempPoint := Points.Items[i] as TPoint;   //jeden "aktuální "si vytáhnu
       if TempPoint.Y = MaxY then begin          //pokud se jejich y-souřadnice rovnají
         //zjistím zda aktuální není více vpravo než doposud zvolený nejlepší
         if TempPoint.X > MaxX then begin
           MaxIndex := i;  //pokud ano, aktuální nastavím jako ten doposud zvolený nejlepší
           MaxY := TempPoint.Y;
           MaxX := TempPoint.X;
         end;
       end;
       //pokud y-souřadnice aktuálního je lepší než doposud zvoleného nejlepšího
       if TempPoint.Y < MaxY then begin
         MaxIndex := i;                          //mám nový nejlepší
         MaxY := TempPoint.Y;                    //nastavim ho
         MaxX := TempPoint.X;
       end;
     end;
     //orotuji řetězec tolikrát (MaxIndex), kolikrát je potřeba
     if MaxIndex > 0 then for i := 0 to MaxIndex do aPattern := StringRotation(aPattern);
     Points.Free;          //uvolním seznam souřadnic bodů
     result := aPattern;  //vrátím výsledek
   end;
```

Obr. 21 Algoritmus zajištění invariance vůči volbě počátečního bodu popisu

Nejprve se vypočtou souřadnice všech bodů objektu (pro jednoduchost je objekt umístěn dostatečně „daleko“ od počátku soustavy souřadnic) a uloží se do indexovaného lineárního spojového seznamu. Dále se zjistí, který bod se nachází nejvíce „vpravo nahoře“ - seznam bodů je v cyklu prohledán a každý bod se porovná s y-souřadnicí bodu uloženého v pomocné proměnné. Pokud je souřadnice menší, uloží se do pomocné proměnné právě zpracovávaný bod seznamu, pokud je y-souřadnice větší, pokračuje se dalším bodem v seznamu. V případě rovnosti dochází k porovnání x-souřadnic obou bodů a pokud je x-souřadnice zpracovávaného bodu větší než x-souřadnice bodu v pomocné proměnné, uloží se zpracovávaný bod do pomocné proměnné. Hodnota pomocné proměnné musí být na začátku algoritmu nastavena na hodnotu vyšší než maximální y-souřadnice všech bodů ze seznamu. Nakonec je provedena rotace vstupního řetězce o počet symbolů rovný indexu nalezeného bodu.

6.2 ZAJIŠTĚNÍ INVARIANCE VŮČI ROTACI

Invarianci vůči rotaci můžeme zajistit použitím diferenciálních primitiv (viz kapitola 4.1.2) pro popis objektu. Popis objektu vytvořený pomocí diferenciálních primitiv je na rozdíl od „obyčejných“ primitiv (které jsou invariantní pouze vůči translaci) invariantní i vůči rotaci a tedy invariancí vůči rotaci se není třeba dále zabývat. Takto vytvořený popis má ovšem při použití pro rozpoznávání deformovaných objektů pomocí deformačních gramatik dvě nevýhody. Ačkoliv je vzniklý popis objektu pomocí diferenciálních primitiv stejně dlouhý jako popis pomocí „klasických“ primitiv, je ale obvykle jednodušší a pro jeho vytvoření je potřeba menší počet primitiv, například u popisu testovacího objektu „stylizovaný dům“:

- ddfffbbbcccaaccbbcccaaaggg, klasická primitiva, n_p=6
- aadaadaabaabacacabaabaadaa, diferenciální primitiva, n_p = 4

Dále jsou u diferenciálních primitiv vždy vyšší váhy, z důvodu samotného principu diferenciálních primitiv, viz část objektu na obr. 22.

Obr. 22a (vlevo) Nedeformovaná část objektu, obr. 22b (vpravo) deformovaná část objektu

- popis nedeformované části objektu, obr. 22a: *cccc*
- popis deformované části objektu, obr. 22b pomocí klasických primitiv: *cehc*, hodnota vah popisující deformací w=2
- popis deformované části objektu, obr. 22b pomocí diferenciálních primitiv: *adcd*, hodnota vah popisující deformací w=3

Vyšší hodnota vah je způsobena, že pro popsání deformace dvou primitiv je nezbytné změnit směr popisu třikrát, což ve spojení s jednodušším popisem ve výsledku znamená menší rozdíly ve váhách mezi objekty (větší podobnost popisu objektů) a tedy vyšší pravděpodobnost nesprávné klasifikace objektu. Ovšem tento způsob zvláště pak ve spojení s automatickou volbou počátečního bodu popisu nabízí výbornou rychlost rozpoznávání.

Pro zajištění maximální možné úspěšnosti klasifikace je tedy výhodnější použít způsob jiný, spočívající v rotaci rozpoznávaného objektu. V případě použití osmi - okolí pro popis objektu je možných variant rotace objektu osm, což lze ještě efektivně analyzovat. Algoritmus pro rotaci objektu je velice jednoduchý, spočívá v transformaci jednotlivých primitiv a tedy získání jiného řetězce. Je naznačen na obr. 23.

```
function ObjectRotation(aString: string): string;
//funkce vrátí popis objektu aString otočený o 45 stupňů
begin
  result := '';                          //vynuluji výsledek
  for i := 1  to aString.Length do begin   //pro každý znak řetězce vykonám:
    case aString[i] of                   //vezmu i-tý symbol řetězce a transformuji ho
      'a': result := result + 'g';       //tedy otočím o 45 stupňů
      'g': result := result + 'd';
      'd': result := result + 'f';
      'f': result := result + 'b';
      'b': result := result + 'h';
      'h': result := result + 'c';
      'c': result := result + 'e';
      'e': result := result + 'a';
      else result := result + aString[i];  //oblouky se otočením nemění
    end;
  end;
end;
```

Obr. 23 Algoritmus rotace objektů

Celkový algoritmus pro zajištění invariance vůči volbě bodu počátku popisu pomocí automatické volby počátku popisu a invariance vůči rotaci pomocí rotace objektu je naznačen na obr. 24.

```
procedure AnalyzeGrammars(aPattern: string);
//provede syntaktickou analýzu řetězce aPattern pro seznam gramatik AnalyzeGrammarList
var rPattern: string;     //pomocná proměnna pro rotaci řetězce
    BestGrm: string;      //pomocná proměnna se jménem nejlepší gramatiky
    Grm: TGrammar;        //ukazatel na gramatiku, která se vkládá do parseru
    MinWGrm, ResGrm: single;     //pomocné proměnné pro vyhledání nejmenší váhy pro jednu
gramatiku
    při zajišťování rotační invariance
    MinW, Res: single;    //pro všechny gramatiky
begin
  //otestuji, zda mám zadané gramatiky k analýze
  //otestuji, zda mám neprázdný vstupní řetězec
  MinWGrm := 500000;          //nastavení pomocné proměnné pro hledání nejlepší gramatiky
  aPattern  :=  NormalizeBeginOfDescription(aPattern,  LineObject.Scale);    //normalizuje  se
počáteční
  bod
  for i := 0 to self.AnalyzeGrammarList.Count - 1 do begin  //prochází se všechny položky
seznamu
  gramatik
    //inicializují se pomocné proměnné pro nalezení nejmenší váhy
```

```
    ResGrm := -1;    //-1 znamená, že nebyla nalezena žádná gramatika, která generuje vstupní
    řetězec
    MinW := 500000;              //takto vysoká váha se v průběhu analýzy určitě nevyskytne
    rPattern := aPattern;        //vezme se řetězec
    for j := 1 to 8 do begin     //pro osm možných natočení objektu – osmiokolí
      Res := par.Early(GrmToOptm, rPattern);  //zjistím Earlyho vzdálenost
      //a zjištěná hodnota se porovná s již nalezenými výsledky
       if Res <> -1 then begin      //pokud aktuální gramatika může generovat rPattern
         if Res < MinW then begin   //porovnají se váhy
           MinW := Res;             //ta lepší se uchová
           ResGrm := Res;           //zde je ukládán nejlepší výsledek
         end;
       end;
       rPattern := ObjectRotation(rPattern);  //objekt se pootočí o 45 stupňů
       rPattern  :=  NormalizeBeginOfDescription(rPattern,  LineObject.Scale);//normalizuje  se
počátek
    end;
    if ResGrm <> -1 then begin  //zda byl nalezen vůbec nějaký parsing
      //výpis výsledku
      //zjištený výsledek se porovná s již zjištěnými výsledky ostatních gramatik
      if ResGrm < MinWGrm then begin
        MinWGrm := ResGrm;  //zapamatuji si nejlepší výsledek
        BestGrm := Grm.Name;
      end;
    end;
  end;
  //výpis výsledků
end;
```

Obr. 24 Celkový algoritmus pro zajištění základních invariancí

Vstupem algoritmu je analyzovaný řetězec *aPattern* a seznam gramatik *AnalyzeGrammarList*. Výstupem algoritmu je výsledek analýzy, která gramatika (pokud vůbec nějaká) je schopna generovat vstupní řetězec s nejmenším součtem vah chybových (deformačních) pravidel.

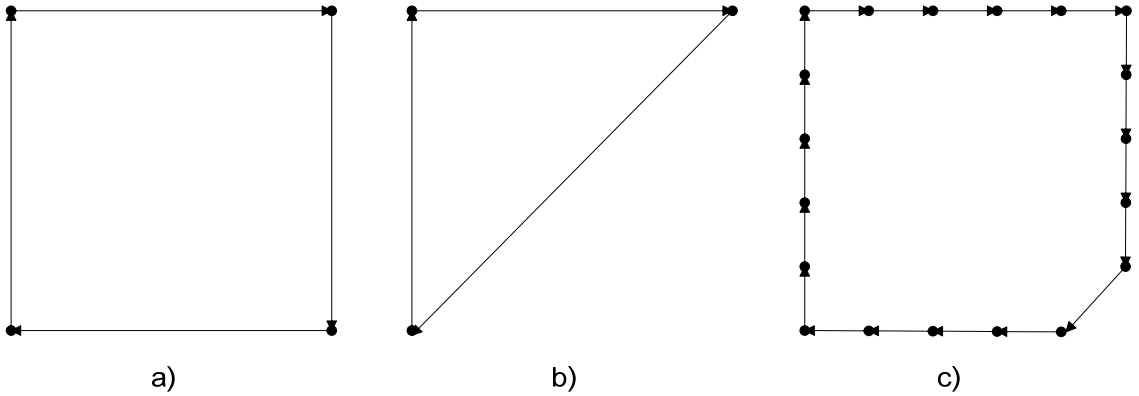
Nejprve otestuji, zda mám zadané gramatiky k analýze, zda mám zadaný neprázdný vstupní řetězec a normalizuji počátek popisu vstupního řetězce *aPattern* pomocí algoritmu uvedeného na obr. 21. Následně se pro každou gramatiku ze seznamu *AnalyzeGrammarList* provede syntaktická analýza všech možných osmi natočení řetězce *aPattern* reprezentujícího rozpoznávaný objekt. Nejnižší zjištěná váha je vybrána a dále se testuje, zda není lepší (menší) než zjištěné váhy ostatních gramatik. Nejmenší hodnota je uchována a vypsána společně s odpovídající gramatikou a údaji o analýze jako výsledek.

6.3 INVARIANCE VŮČI ZMĚNĚ MĚŘÍTKA

Invariance vůči změně měřítka zajistí korektní rozpoznání objektů stejného tvaru, avšak různé velikosti. Tato invariance obvykle nebývá požadována a popis objektů pomocí primitiv obvykle nebere v potaz jakékoliv měřítko primitiv, je vytvořen pouze z celých hran daných směrů viz obr. 25a.

Obr. 25a (vlevo) Popis objektu pomocí primitiv bez zvoleného měřítka a obr. 25b (vpravo) se zvoleným měřítkem

Ovšem u analýzy deformačních gramatik je způsob neberoucí v potaz měřítko primitiv nevhodný. Velikost vlastních deformaci tvaru bývá oproti velikosti objektu nevýznamná. Proto při velmi malé délce slova popisujícího objekty dochází již při aplikaci několika málo deformačních pravidel ke značné deformaci objektu, která může vést ke vzniku popisu jiného objektu, jak je ilustrováno na obr. 26, kde na obr. 26a je objekt – čtverec popsán pomocí primitiv bez měřítka a již po aplikaci dvou deformačních pravidel se z objektu stává nový objekt – trojúhelník, obr. 26b. Na obr. 26c je stejný objekt jako na obr. 26a popsán pomocí primitiv s měřítkem a jak je vidět, aplikace deformačních pravidel umožňuje popsat deformaci objektu (například chyba při snímání rohu objektu), aniž by došlo ke vzniku úplně jiného objektu.



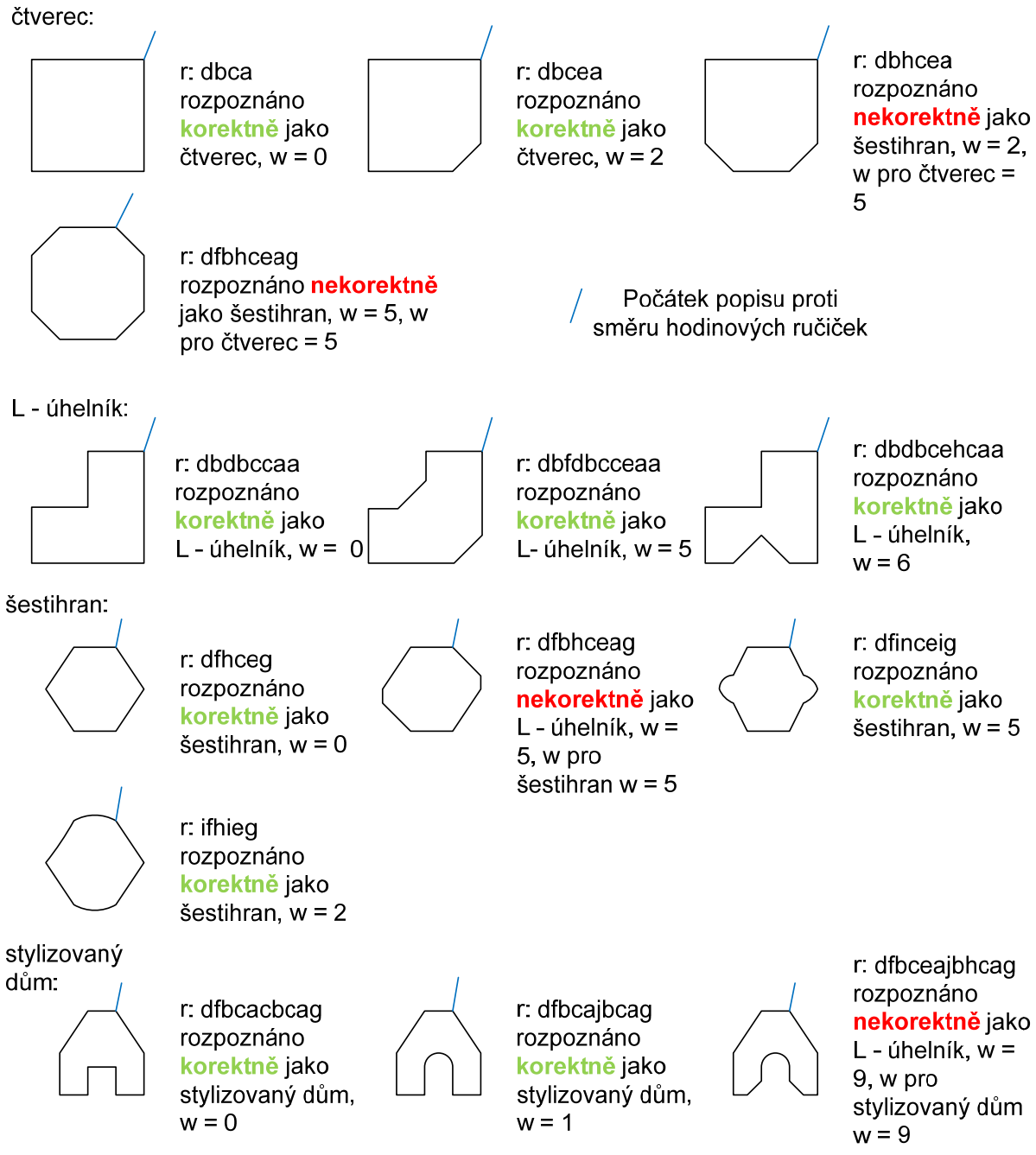
Obr. 26 Měřítko primitiv

Při použití deformačních gramatik je tedy nutno dbát na výběr vhodné délky slova a měřítka primitiv popisujících rozpoznávaný objekt. Je nezbytné volit slova dostatečné délky vzhledem k velikosti a množství deformací. Při dostatečné délce slov tato metoda funguje korektně, nedochází k chybné identifikaci objektu. Tato metoda je též díky této skutečnosti vhodná pro rozpoznávání objektů v měřítku. Chybná identifikace objektů z důvodu příliš malé délky slov popisujících objekty je znázorněna na obr. 27.

Při zajišťování invariance vůči změně měřítka záleží na vlastní gramatice popisující objekt, zda daná gramatika dokáže generovat stejné objekty různé velikosti (viz například v kapitole 4.3 typ 2) nebo jen objekt jedné velikosti (viz například v kapitole 4.3 typ 1 a typ 3).

6.4 ZPRACOVÁNÍ VNITŘNÍCH HRAN OBJEKTU

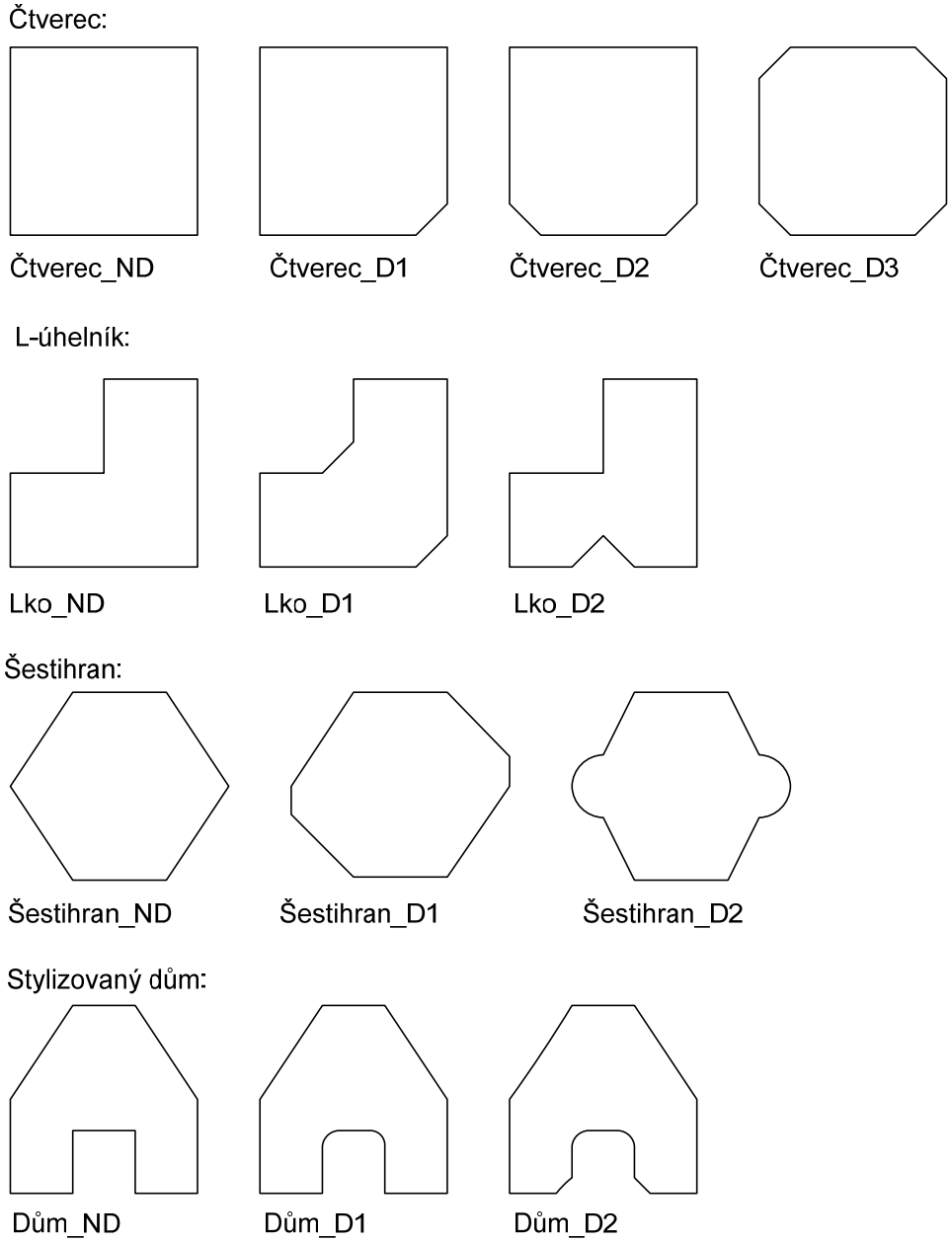
Některé objekty mají též i svoji vnitřní hranu (například matice) a je tedy nezbytné, aby metoda pro jejich rozpoznání dokázala vnitřní hrany zpracovat. U syntaktických metod je tato úprava velice jednoduchá. Stačí do datové struktury popisující objekt vložit odkaz na popis vnitřní hrany, která se dále popíše a analyzuje stejně jako vnější hrana objektu. S popisem objektu se pracuje beze změn, včetně veškerého zajištění invariancí, pracuje se s oběma popisy hran společně. Dochází k úpravě vyhodnocování analýzy, do které je nutné zahrnout zpracování vnitřní hrany. Při analýze deformační gramatiky s vnitřními hranami taktéž nedochází k žádným změnám.



Obr. 27 Chybné rozpoznání objektů při zvolené malé délce slova

7 VÝSLEDKY

Všechny výsledky popisovaných algoritmů uvedených v této kapitole byly získány na konfiguraci 1.8GHz single core Athlon 64 s 3072MB RAM a operačním systémem Windows Server 2003 32bit. Pro zjištění úspěšnosti klasifikace objektů byla použita sada čtyř testovacích objektů, které se podobají dvourozměrným obrazům reálných předmětů a zároveň jsou vhodné pro generování několika jejich zdeformovaných variant. Záměrně byly zvoleny některé objekty tvarově blízké a především s podobnými deformovanými tvary. Ačkoliv je počet testovacích objektů malý, ke každému testovacímu objektu ovšem přísluší několik jeho deformovaných verzí, tudíž celkový počet objektů k otestování úspěšnosti rozpoznávání je více než dostatečný. Testovací objekty a jejich deformované varianty jsou na obr. 28.



Obr. 28 Testovací objekty

Pro testy vlivu optimalizace na rychlost algoritmu byly použity stejné testovací objekty, pouze o větší délce slova (jiného měřítka primitiv).

7.1 VÝSLEDKY ÚSPĚŠNOSTI ROZPOZNÁVÁNÍ OBJEKTŮ

Pro čtyři gramatiky generující testovací objekty z obr. 28 se postupně analyzují jednotlivé řetězce odpovídající testovacím objektům a jejich deformovaným variantám. Invariance vůči rotaci je v testovacím prostředí zajištěna pomocí rotace objektů (metoda popsaná v kapitole 6.2) a invariance vůči volbě bodu počátku popisu je zajištěna pomocí automatické volby bodu počátku popisu (metoda popsaná v kapitole 6.1). Délka slov popisujících objekty je volena s přihlédnutím na velikost deformací objektů. Pro porovnání jsou pro objekt „čtverec_ND“ uvedeny výsledky pro analýzu bez i s e-pravidly. Výsledky rozpoznávání jsou uvedeny v tab. 11. Výsledky rychlosti algoritmů jsou pouze orientační, protože implementovaný algoritmus obsahuje množství ladících informací, které dobu analýzy znatelně prodlužují.

Objekt	Řetězec	Délka	E-pravidla	Rozpoznáno	Váha	Čas [s]
Čtverec						
Čtverec_ND	ddddddbbbbbbccccccaaaaaa	24	NE	ANO	0	1.281
Čtverec_ND	ddddddbbbbbbccccccaaaaaa	24	ANO	ANO	0	1.375
Čtverec_D1	ddddddbbbbbbcccccceaaaaa	24	NE	ANO	2	1.281
Čtverec_D1	ddddddbbbbbbccccccaaaaa	23	ANO	ANO	2	1.390
Čtverec_D2	ddddddbbbbbbhhccccceaaaaa	24	NE	ANO	4	1.515
Čtverec_D2	ddddddbbbbbbhccccceaaaaa	22	ANO	ANO	4	1.406
Čtverec_D3	ggddddffbbbbbhhccccceaaaa	24	NE	ANO	8	1.422
Čtverec_D3	ddddfbbbbhccccceaaaag	20	ANO	ANO	10	1.548
L-úhelník						
Lko_ND	dddbbbddddbbbccccccaaaaaa	24	NE	ANO	0	1.235
Lko_D1	dddbbffddbbbccccccceaaaaa	24	NE	ANO	4	1.218
Lko_D2	dddbbbddddbbbccehccaaaaaa	24	NE	ANO	2	1.265
Šestihran						
Šestihran_ND	dddfffhhhccceeeggg	18	NE	ANO	0	1.078
Šestihran_D1	fffbhhccceeeaggddd	18	NE	ANO	2	0.953
Šestihran_D2	fffiihhhccceeeiigggddd	22	NE	ANO	4	1.015
Stylizovaný dům						
Dům_ND	ddfffbbbcccaaccbbcccaaaggg	26	NE	ANO	0	1.469
Dům_D1	dddbbffddbbbccccccceaaaaa	26	NE	ANO	2	1.312
Dům_D1a	ddddddbbbbbbccccccceaaaaa	27	ANO	ANO	2	1.719
Dům_D2	ddfffbbbccceaajcjbbhcccaaaggg	29	NE	ANO	4	1.688

Tab. 11 Výsledky úspěšnosti rozpoznávání objektů

7.2 VÝSLEDKY ROZPOZNÁVÁNÍ OBJEKTŮ POPSANÝCH POMOCÍ DIFERENCIÁLNÍCH PRIMITIV

Pro ilustraci jsou zde uvedeny i výsledky rozpoznávání deformovaných objektů popsaných za pomoci diferenciálních primitiv. Délka řetězců použitých pro popis testovacích objektů *n* je 24 nebo více znaků, uváděné časy analýzy jsou pouze orientační. Neoptimalizovaným časem je míněn čas analýzy bez optimalizací parseru popsaných v kapitole 5. Výsledky rozpoznávání jsou uvedeny v tab. 12.

Objekt	Rozpoznáno, váha	Čas [s]	Neoptimalizovaný čas [s]
Čtverec_ND	ANO, $w_D = 0$	2.062	5.500
Čtverec_D1	ANO, $w_D = 3$	3.563	5.031
Čtverec_D2	ANO, $w_D = 6$	3.594	5.063
Dům_ND	ANO, $w_D = 0$	5.390	6.828
Dům_D1	ANO, $w_D = 3$	5.516	6.781
Dům D2	ANO, $w_D = 9$	5.547	7.453
L-úhelník_ND	ANO, $w_D = 0$	2.703	5.063
L-úhelník_D1	NE, rozpoznáno jako čtverec, $w_D = 6$	4.453	4.859
L-úhelník_D2	ANO, $w_D = 3$	4.046	4.797
Šestihran_ND	ANO, $w_D = 0$	1.453	1.766

Tab. 12 Rozpoznávání objektů popsaných pomocí diferenciálních primitiv

7.3 VÝSLEDKY OPTIMALIZACÍ EARLYHO PARSERU

Rychlost Earlyho parseru použitého pro analýzu deformační gramatiky je v celé řadě aplikací zabývající se rozpoznáváním klíčová. Jeho rychlost je příznivě ovlivněna popsanými optimalizacemi. Protože implementovaný algoritmus obsahuje množství ladících informací, které dobu analýzy znatelně prodlužují, jsou ve výsledcích uváděny počty Earlyho stavů, nejen doby analýzy. Invariance vůči volbě počátečního bodu popisu a invariance vůči rotaci nejsou pro jednoduchost a zkreslení těchto výsledků zahrnuty.

Nejjednodušší z nich je optimalizace gramatiky pro konkrétní vstupní slovo. Výsledky jsou uvedeny v tab. 13.

	Bez optimalizace			S optimalizací			Optimalizace
Objekt	Čas[s]	Stavů C/P	C+P	Čas[s]	Stavů C/P	C+P	počtu stavů
Čtverec	0.282	3101/3020	6121	0.171	3101/1820	4921	19.6%
Čtverec D1(w_D=2)	0.291	3101/3020	6121	0.187	3101/2020	5121	16.3%
L-úhelník	0.250	3431/3130	6561	0.218	3431/1870	5301	19.2%
L-úhelník D1(w_D=4)	0.266	3431/3130	6561	0.234	3431/2500	5931	9.6%
Dům	0.453	5461/4428	9889	0.343	5461/3116	8577	13.3%
Dům D1(w_D=8)	0.469	5461/4428	9889	0.391	5461/3772	9233	6.6%

Tab. 13 Optimalizace gramatiky pro konkrétní vstupní slovo

Příznivý vliv optimalizace gramatiky pro konkrétní vstupní slovo klesá se členitostí objektu a s jeho deformací. Prakticky lze dosáhnout přibližně 0-25% zmenšení počtu Earlyho stavů při zanedbatelné náročnosti vlastní optimalizace.

Další z popisovaných optimalizací je filtrování předpovědí nemožných stavů podle aktuálního vstupního symbolu v operaci predikce. Výsledky jsou uvedeny v tab. 14.

	Bez optimalizace			S optimalizací			Optimalizace
Objekt	Čas[s]	Stavů C/P	C+P	Čas[s]	Stavů C/P	C+P	počtu stavů
Čtverec	0.282	3101/3020	6121	0.094	1541/2020	3561	41.8%
Čtverec D1(w_D=2)	0.291	3101/3020	6121	0.099	1587/2050	3637	40.6%
L-úhelník	0.250	3431/3130	6561	0.109	1881/2335	4216	35.7%
L-úhelník D1(w_D=4)	0.266	3431/3130	6561	0.110	1897/2420	4317	34.2%
Dům	0.453	5461/4428	9889	0.203	2917/3182	5899	40.3%
Dům D1(w_D=8)	0.469	5461/4428	9889	0.125	1762/2579	4341	56.1%

Tab. 14 Filtrování předpovědí nemožných stavů podle aktuálního vstupního symbolu

Vliv optimalizace filtrováním předpovědí nemožných stavů lze jen těžko přesně vyjádřit, ovšem je významný. Lze dosáhnout až 60% snížení počtu Earlyho stavů.

Poslední z popisovaných optimalizací je zamezení generování zbytečných stavů v závislosti na následujícím vstupním symbolu. Výsledky jsou uvedeny v tab. 15.

	Bez optimalizace			S optimalizací			Optimalizace
Objekt	Čas[s]	Stavů C/P	C+P	Čas[s]	Stavů C/P	C+P	počtu stavů
Čtverec	0.282	3101/3020	6121	0.172	3101/1220	4321	29.4%
Čtverec D1(w_D=2)	0.291	3101/3020	6121	0.172	3101/1220	4321	29.4%
L-úhelník	0.250	3431/3130	6561	0.187	3431/1240	4671	28.8%
L-úhelník D1(w_D=4)	0.266	3431/3130	6561	0.188	3431/1240	4671	28.8%
Dům	0.453	5461/4428	9889	0.265	5461/1476	6937	29.8%
Dům D1(w_D=8)	0.469	5461/4428	9889	0.266	5461/1476	6937	29.8%

Tab. 15 Zamezení generování zbytečných stavů v závislosti na následujícím vstupním symbolu

Vliv optimalizace zamezením generování zbytečných stavů v závislosti na následujícím vstupním symbolu je též obtížné přesně vyjádřit. Pro konkrétní návrh deformační gramatiky se ukazuje, že lze dosáhnut přibližně 30% snížení počtu Earlyho stavů.

Celkové výsledky všech popsaných optimalizaci jsou uvedeny v tab. 16.

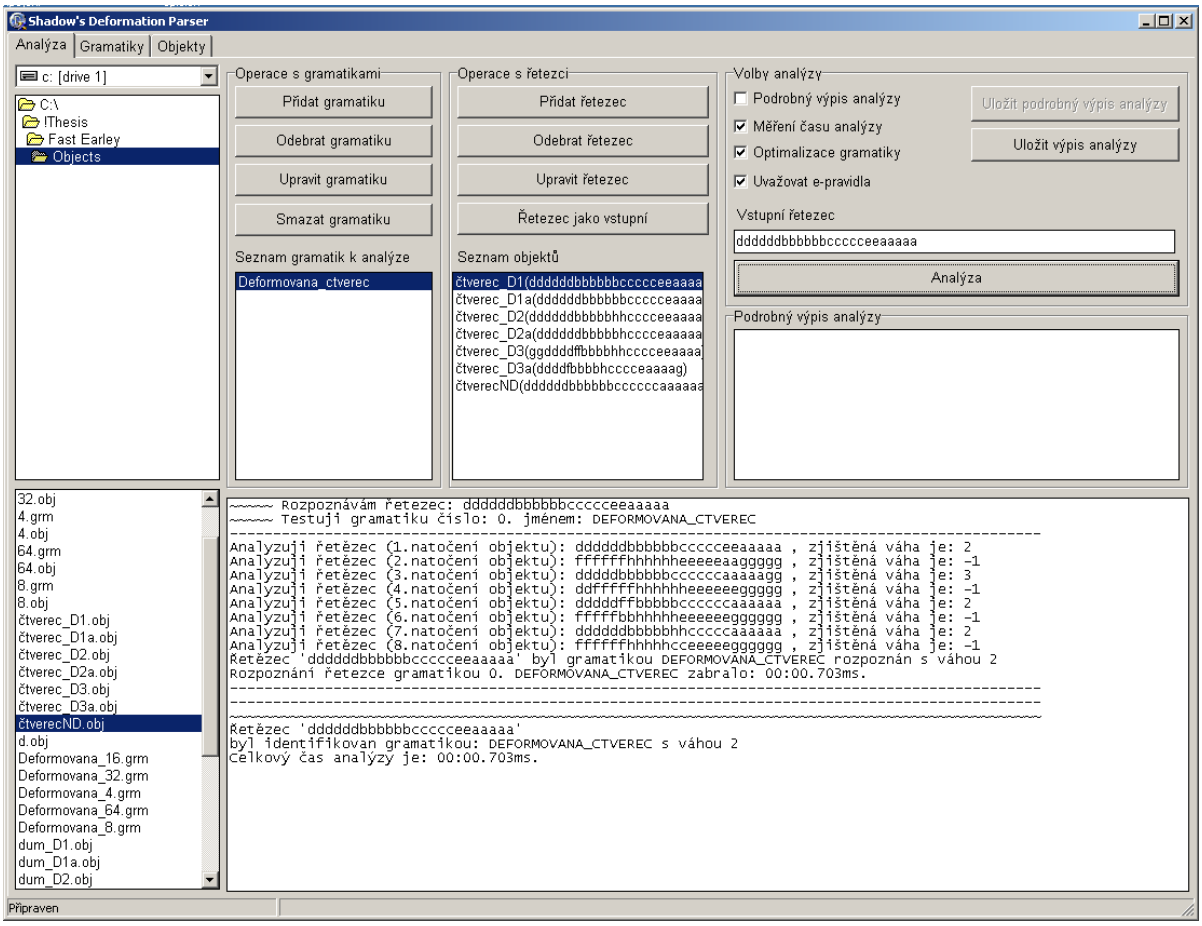
	Bez optimalizace			S optimalizací			Optimalizace
Objekt	Čas[s]	Stavů C/P	C+P	Čas[s]	Stavů C/P	C+P	počtu stavů
Čtverec	0.282	3101/3020	6121	0.062	1541/850	2391	60.9%
Čtverec D1(w_D=2)	0.291	3101/3020	6121	0.063	1587/864	2451	60.0%
L-úhelník	0.250	3431/3130	6561	0.062	1881/941	2822	56.9%
L-úhelník D1(w_D=4)	0.266	3431/3130	6561	0.078	1897/962	2859	56.4%
Dům	0.453	5461/4428	9889	0.125	2917/1148	4065	58.9%
Dům D1(w_D=8)	0.469	5461/4428	9889	0.078	1762/921	2683	72.8%

Tab. 16 Celkové výsledky

Z tab. 16 je patrno, že celkové výsledky nejsou „součtem“ dílčích výsledků jednotlivých optimalizací, ale že jednotlivé optimalizace se navzájem ovlivňují a celkové výsledky jsou proto nižší. Běžně je dosahováno až 60% snížení počtu Earlyho stavů. Nižší počet stavů při analýze příznivě ovlivňuje rychlost analýzy. Dále doba analýzy závisí na délce analyzovaného slova, se kterou roste exponenciálně. Pro zvýraznění výsledků jsou volena velmi dlouhá slova ($n > 64$).

8 POPIS TESTOVACÍHO PROSTŘEDÍ A JEHO ARCHITEKTURY

Pro otestování možností deformačních gramatik bylo vytvořeno testovací prostředí (obr. 29), které obsahuje všechny potřebné funkce nutné pro ověření analýzy náhodně deformovaných objektů. Pro vlastní rozpoznání objektů je v něm implementován modifikovaný Earlyho algoritmus z kapitoly 5. Vytvořený algoritmus mimo jiné obsahuje filtrování předpovědí nemožných stavů podle aktuálního vstupního symbolu, zamezení generování zbytečných stavů v závislosti na následujícím vstupním symbolu a optimalizace gramatiky pro konkrétní vstupní slovo. Použita jsou klasická primitiva z kapitoly 4.1.1 (též viz příloha A). Diferenciální primitiva (kapitola 4.1.2, příloha B) nejsou zahrnuta. Byla testována pouze v průběhu vývoje testovacího prostředí a z důvodů popsaných v kapitole 6.2 byla zavržena.



Obr. 29 Vytvořené testovací prostředí, záložka Analýza

Vstupem analýzy testovacího prostředí jsou řetězce popisující objekty k rozpoznání. Ačkoliv je oblast image preprocesingu a detekce hran důležitou součástí jakýchkoliv strukturálních metod pro rozpoznávání objektů, je to poměrně široká oblast, která je dostatečně dobře popsaná v nejrůznějších pramenech (například v [13], [31], [7], [36], [35], [30], [8]), proto se s ní tato práce nezabývá a vstupem implementovaného algoritmu jsou řetězce odpovídající objektům k rozpoznání a nikoliv „obrázky“ objektů k rozpoznání. Řetězce lze vizuálně vytvořit a uložit na disk v editoru objektů, který je součástí testovacího

prostředí. Testovací prostředí obsahuje funkci, která pro nakreslený objekt automaticky vytvoří gramatiku, jež je ho schopná generovat. Nejkratší řetězec, který je prostředí schopno rozpoznat, má délku dva znaky.

Dále jsou vstupem analýzy gramatiky popisující třídy objektů k rozpoznání. Testovací prostředí obsahuje editor gramatik pro pohodlnou práci s nimi. Z již vytvořených gramatik lze vytvořit seznam gramatik sloužící pro rozpoznávání objektů.

8.1 ARCHITEKTURA TESTOVACÍHO PROSTŘEDÍ

Testovací prostředí bylo implementováno v jazyce Borland Dephi for .NET 2007 s využitím frameworku .NET 3.0, který je též potřeba pro vlastní běh testovacího prostředí. Testovací prostředí bylo vyvíjeno s uživatelským rozhraním v českém jazyce, bez možnosti lokalizace pomocí *resources* souborů. Aplikace je vhodná pro rozlišení 1280x1024 a vyšší. Uživatelské rozhraní je řešeno záložkově (tři záložky) se stavovým řádkem a bez menu. Ošetření chyb v aplikaci se omezuje pouze na jejich detekci a následné vypsání chybové zprávy. Při vývoji testovacího prostředí byly použity pouze standardně dodávané komponenty.
Testovací prostředí se skládá z následujících tří uživatelských jednotek:

- jednotka Engine – obsahující definice třídy Engine, která je nezbytná pro funkci testovacího prostředí
- jednotka DefGrm – obsahující definice pravidel, stavu, gramatiky a parseru
- jednotka mFe – hlavní jednotka obsahující definici formuláře aplikace a jejího uživatelského rozhraní

V adresáři aplikace jsou umístěny dva podadresáře *Objects* a *Grammars* pro ukládání testovacích objektů a gramatik vytvořených v testovacím prostředí. Konfigurační soubor je uložen v adresáři aplikace.

8.2 DATOVÉ STRUKTURY TESTOVACÍHO PROSTŘEDÍ

Jednotka ***DefGrm*** obsahuje následující datové struktury:

- TRule – záznam reprezentující jedno pravidlo gramatiky.
- TState – záznam reprezentující jeden stav Earleyho parseru.
- TGrammar – třída popisující jednu gramatiku, její terminály, neterminály, pravidla, jméno, počáteční symbol, metody pro vytváření deformační gramatiky, ukládání do a nahrávání gramatiky z textového souboru.
- TParser – třída zajišťující rozpoznání objektů, obsahující seznam gramatik a implementaci modifikovaného Earlyho parseru schopného rozpoznávání deformačních gramatik.

Jednotka ***Engine*** obsahuje následující datové struktury:

- TLine – záznam reprezentující jednu hranu objektu.
- TLineObject – třída pro reprezentaci jednoho objektu, obsahující seznam hran, ukazatel na první bod objektu, měřítko primitiv, jméno objektu, metody pro ukládání do a nahrávání z textového souboru.

- TMainEngine – třída nezbytná pro funkci testovacího prostředí, obsahující seznam gramatik k analýze, seznam řetězců k rozpoznání, výpis analýzy, detailní výpis analýzy, instanci třídy TGrammar pro obsluhu záložky „Gramatiky“ testovacího prostředí, instanci třídy TLineObject pro obsluhu záložky „Objekty“ testovacího prostředí, pomocné proměnné pro ovládání testovacího prostředí, *.ini soubor aplikace, metody pro uchování a obnovení nastavení testovacího prostředí, metody pro zajištění invariance vůči bodu počátku popisu a rotaci a metodu obsluhující vlastní rozpoznávání objektů.

Jednotka ***mFE*** obsahuje následující datové struktury:

- TMainForm - potomek třídy TForm obsahující definici formuláře zajišťujícího obsluhu uživatelského rozhraní testovacího prostředí.

Jednotka ***mFE*** obsahuje následující globální proměnné:

- MainForm – instance třídy TMainForm obsahující hlavní formulář aplikace.
- Engine - instance třídy TMainEngine zajišťující vlastní běh aplikace.

8.3 ALGORITMY IMPLEMENTOVANÉ V TESTOVACÍM PROSTŘEDÍ

V této kapitole jsou pomocí pseudokódu popsány nejdůležitější algoritmy implementované v testovacím prostředí.

8.3.1 Generování gramatiky popisující objekty

Procedura Create_Grammar (obr. 30) vytváří ze zadaného řetězce odpovídajícího rozpoznávanému objektu gramatiku, jež je schopna generovat jeho popis.

```
procedure Create_Grammar(řetězec_objektu: string);
var s, N: string;
    TempRule: TRule;
begin
  Engine.Grammar := TGrammar.Create(InputBox('Jméno gramatiky', 'Zadejte jméno nově vytvářené
  gramatiky:',''));
  s := řetězec_objektu;
  //vytvořím terminály (malá pismena) a neterminály (velká písmena)
  for i := 1 to s.Length do begin
    N := s[i];
    if not Engine.Grammar.IsInTerminals(s[i]) then Engine.Grammar.Terminals.Add(N);
    s[i] := UpCase(s[i]);
    N := s[i];
    if not Engine.Grammar.IsInNeTerminals(s[i]) then Engine.Grammar.NeTerminals.Add(N);
  end;
  s := NormalizeBeginOfDescription(s, Engine.LineObject.Scale);
  TempRule.LHS := 'S';
  TempRule.RHS := s;
  TempRule.Weigth := 0;
```

```
    Engine.Grammar.Rules.Add(TempRule);   //vložím počáteční pravidlo S -> řetězec_objektu,
váha 0
    //vložím pravidla popisující přepis neterminálu na terminály
    for i := 0 to Engine.Grammar.Neterminals.Count - 1 do begin
      N := Engine.Grammar.Neterminals.Items[i] as string;
      TempRule.LHS := N;
      TempRule.RHS := LowerCase(N);
      TempRule.Weigth := 0;
      Engine.Grammar.Rules.Add(TempRule);
    end;
    Engine.Grammar.InitSymbol := 'S';
    Engine.Grammar.Neterminals.Add(s);
  end;
```

Obr. 30 Algoritmus procedury Create_Grammar

8.3.2 Vlastní analýza řetězce pomocí modifikovaného Earlyho parseru

Funkce (obr. 31) vrací výsledek analýzy vstupního řetězce *aInputString* a gramatiky *aGrammar* modifikovaným Earlyho parserem. Dále je vstupem procedury váha případné aplikace e-pravidel *aWDelete*. Z prostorových důvodů nejsou operace Prediction, Complete a Scan dále rozváděny. K akumulaci vah deformačních pravidel dochází v operaci Complete.

```
function TParser.Early(aGrammar: TGrammar; aInputString: string; aWDelete: word): integer;
var TempR: TRule;                   //pomocná proměnná pro práci s pravidly (inicializace)
    TempS, TempSNew: TState;        //pomocná proměnna pro práci se stavy z SListu
    T: string;                      //proměnná reprezentující jeden terminál
Begin
  result := -1;  //pokud nebylo identifikováno, vrátím -1
  NSign := 'x';   //v nulté úrovni seznamu ještě nezpracovávám žádné primitivum a nic se
neprořezává
  NextSign := aInputString[1];  //při vytváření nulté úrovně seznamu je následující symbol
  //vstupního řetězce první symbol vstupního řetězce
  CreateLevel(0);  //vytvořím SList[0] a vložím počáteční pravidlo/la
  for i := 0 to aGrammar.Rules.Count - 1 do begin  //procházím všechna pravidla gramatiky
    TempR := aGrammar.Rules.Items[i] as TRule;     //vždy si jedno vytáhnu
    if TempR.LHS = aGrammar.InitSymbol then SListP[0].Add(TempSNew);  //a otestuji, jestli na
    //jeho pravé straně není počáteční symbol a pokud, je vytvořím z pravidla Earlyho stav
    // a s váhou 0 vložím do nulté úrovně seznamu SListP
  end;
  Prediction(aGrammar, 0, aWDelete); //predikce S0 - rozvíjím vložená počáteční pravidla
  //vlastní jádro algoritmu
  for i := 1 to aInputString.Length do begin   //cyklus zpracuje všechny znaky vstupního
řetězce
    CreateLevel(i);  //vytvořím i-tou úroveň seznamu
    //určím NextSign podle pozice v řetězci
    if i = aInputString.Length then NextSign := 'x'//když už jsem na konci, beru nějaký
    //nesmysl
                              else NextSign := aInputString[i + 1]; //jinak následující znak
    T := aInputString[i];                   //vezmu i-tý znak vstupního řetězce
    NSign := GetNSign(aInputString[i]);     //zjistím následující nepřípustný symbol
    Scan(SListP[i-1],T, i);  //provedu operaci scan
```

```
    Complete(i);             //provedu operaci complete
   //provedu operaci predict pro úroveň i, aGrammar předávám kvůli pravidlům, pro poslední
   //symbol vstupního řetězce je operace predikce zbytečná
    if i <> aInputString.Length then Prediction(aGrammar, i, aWDelete);
  end;
  //test úspěšnosti - procházím všechny kompletní stavy poslední úrovně
  for i := 0 to SListC[aInputString.Length].Count - 1 do begin
    TempS := SListC[aInputString.Length].Items[i] as TState;     //jeden si vytáhnu
    //a otestuji, zda LHS je počáteční symbol a stav je zkompletován (pro jistotu), jako
    // výsledek vrátím váhu
    if (TempS.LHS=aGrammar.InitSymbol) And (TempS.ToParse='' ) then result := TempS.Weight;
  end;
end;
```

Obr. 31 Algoritmus modifikovaného Earlyho parseru

8.3.3 Vytváření deformační gramatiky

Funkce vytvoří a vrátí ukazatel na deformační gramatiku. Vstupní parametry jsou váhy deformačních pravidel. Váha pro e-pravidla není zadána z důvodů nevytváření e-pravidel, která jsou zpracována v samotném parseru pomocí seznamu nulovatelných terminálů (viz kapitola 5.5.6).

```
function TGrammar.DoDeformationGrammar(aWInsert, aWReplace: word): TGrammar;
//aWInsert - váha vkládání, aWReplace - váha nahrazení
var TempR, NewRule: TRule;   //pomocná proměmná pro vytváření a vkládání jednoho pravidla
    N, T: string;  //pomocné proměnné reprezentující jeden neterminál/terminál
Begin
  //vytvořím novou prázdnou gramatiku a nastavim její jméno
  result := TGrammar.Create('Deformovana_' + self.Name);
  result.InitSymbol := self.InitSymbol; //nastavím počáteční symbol stejný jako u původní
gramatiky
     //Krok 1 - vytvořím seznam terminálů a neterminálů a vložím terminály
     result.Neterminals := TList.Create; result.Terminals := TList.Create;
     N := 'a'; result.Terminals.Add(N); N := 'b'; result.Terminals.Add(N);
     N := 'c'; result.Terminals.Add(N); N := 'd'; result.Terminals.Add(N);
     N := 'e'; result.Terminals.Add(N); N := 'f'; result.Terminals.Add(N);
     N := 'g'; result.Terminals.Add(N);  N := 'h'; result.Terminals.Add(N);
     N := 'i'; result.Terminals.Add(N);  N := 'j'; result.Terminals.Add(N);
     //nyní vytvořím neterminály nové deformační gramatiky, nejprve původní překopíruji
     for i := 0 to self.Neterminals.Count - 1 do begin
       N := Neterminals.Items[i] as string;
       result.Neterminals.Add(N);
     end;
     //a vytvořím deformované neterminály z terminálu + symbol '_',označující def. neterminál
     for i := 0 to result.Terminals.Count - 1 do begin
       N := result.Terminals.Items[i] as string;
       if N <> self.fInitSymbol then result.Neterminals.Add('_' + N);
     end;

     //Krok 2 -  překopíruji a upravím pravidla – náhrada nedeformovaných terminálů v RHS
     deformovanými terminály
```

```
  for i := 0 to self.Rules.Count - 1 do begin
    TempR := Rules.Items[i] as TRule;
    NewRule.LHS := TempR.LHS;
    NewRule.RHS := '';
    NewRule.Weigth := 0;
    for j := 1 to TempR.RHS.Length do begin
      N := TempR.RHS[j];
      if IsInTerminals(N) then NewRule.RHS := NewRule.RHS + '_';
      NewRule.RHS := NewRule.RHS + N;
    end;
    result.Rules.Add(NewRule);
  end;
  //Krok 3
  //3c
  for i := 0 to result.Terminals.Count - 1 do begin
    T := result.Terminals.Items[i] as string;
    TempR.LHS := '_' + T;
    TempR.RHS  := T;
    TempR.Weigth := 0;
    result.Rules.Add(TempR);
  end;
  //3d
  for i := 0 to result.NeTerminals.Count - 1 do begin
    N := result.NeTerminals.Items[i] as string;
    if N.StartsWith('_') then begin
      for j := 0 to result.Terminals.Count - 1 do begin
        T := result.Terminals.Items[j] as string;
        if N[2] <> T then begin
          TempR.LHS := N;
          TempR.RHS := T;
          TempR.Weigth := aWInsert;
          result.Rules.Add(TempR);
        end;
      end;
    end;
  end;
  //3f
  for i := 0 to result.NeTerminals.Count - 1 do begin
    N := result.NeTerminals.Items[i] as string;
    if N.StartsWith('_') then begin
      for j := 0 to result.Terminals.Count - 1 do begin
        T := result.Terminals.Items[j] as string;
        TempR.LHS := N;
        TempR.RHS := T + N;
        TempR.Weigth := awInsert;
        result.Rules.Add(TempR);
      end;
    end;
  end;
end;
```

Obr. 33 Algoritmus generování deformační gramatiky

9 ZÁVĚR A ZHODNOCENÍ VÝSLEDKŮ DEFORMAČNÍCH GRAMATIK

Navržená deformační gramatika spolehlivě generuje všechny možné varianty deformace objektu, a tedy umožňuje použít některou z metod syntaktické analýzy i pro náhodné deformace objektů. Deformační gramatiky mají potenciál rozpoznat jakýkoliv deformovaný objekt, záleží pouze na přesnosti jeho popisu. Oproti metodám pro zjištění vzdálenosti mezi atributovými popisy obrazů je výpočtově (tedy i časově) náročnější, její efektivita je závislá na použitém parseru a efektivitě jeho implementace, která je významně složitější než implementace metod pro zjištění vzdálenosti mezi atributovými popisy obrazů, které pouze využívají „nějakou metriku“. Deformační gramatiky využívají veškeré dostupné znalosti o cílech klasifikace, tedy vlastní gramatiku popisující objekt a tím jsou oproti metodám pro zjištění vzdálenosti mezi atributovými popisy obrazů informovanější. Na rozdíl od nich při správném použití neprodukují chybně rozpoznané objekty, což je největší výhoda této metody.

Při použití je pouze nutno dbát na výběr vhodné délky slova popisujícího rozpoznávaný objekt, jak je podrobně popsáno v kapitole 6.3. Deformační gramatiky mají potenciál objekt rozpoznat dokonale, ovšem taková analýza by byla značně pomalá. Proto je vždy potřeba zvolit vhodný kompromis mezi rychlostí a přesností popisu (měřítko primitiv a tedy následně délka řetězce) rozpoznávaných objektů, zajišťující korektní rozpoznání objektů. Z těchto důvodu je tato metoda zvláště vhodná pro rozpoznávání objektů v měřítku.

Earlyho parser byl pro analýzu deformační gramatiky zvolen, protože nepotřebuje parsovací tabulky, které je obtížné vytvořit. Patrně nejefektivnější z popsaných parserů je LRE(k) parser, který vykazuje potenciál zrychlit analýzu o více než polovinu.

Velký vliv na celkovou rychlost této metody rozpoznávání mají způsoby zajištění invariancí. Byly testovány dva způsoby zajištění invariancí, první za pomocí diferenciálních primitiv pro zajištění invariance vůči rotaci a rotace řetězce pro zajištění invariance vůči volbě bodu počátku popisu. Druhý způsob kombinoval rotaci objektu pro zajištění invariance vůči rotaci a automatickou volbu bodu počátku popisu pro zajištění invariance vůči volbě bodu počátku popisu.

Z výsledků je patrno, že ačkoliv diferenciální primitiva jsou nejrychlejším způsobem pro zajištění invariance vůči rotaci, mohou někdy nepřímo vést k chybné identifikaci objektu, zapříčiněné jednodušším strukturálním popisem objektu a menšími rozdíly v součtu vah deformačních pravidel. Druhý popsaný způsob rotace objektů je použitelný pouze u jednoduchých primitiv (typicky osmiokolí), u složitějších primitiv s více možnostmi natočení dochází k přílišnému zvýšení časové složitosti výpočtu. Tento způsob též zachovává maximální možnou míru úspěšnosti klasifikace objektů.

Rotace řetězce pro zajištění invariance vůči volbě bodu počátku popisu je velice jednoduchý způsob, který ale u dlouhých řetězců významně zvyšuje celkovou časovou složitost analýzy, protože syntaktická analýza se musí pro rozpoznání jednoho objektu provést tolikrát, kolik znaků je délka řetězce. Z tohoto důvodu je automatická volba bodu počátku popisu, kde stačí provést syntaktickou analýzu řetězce pouze jednou, značně efektivnější a přitom vlastní normalizace řetězce je jednoduchá a rychlá.

Invarianci vůči změně měřítka objektu je možné zajistit pomocí volby měřítka (délky) primitiv a zápisu vlastní gramatiky popisující daný objekt. U rozpoznávání deformovaných objektů pomocí deformačních gramatik je k zajištění správné klasifikace objektů tato vhodná volba měřítka primitiv zcela nezbytná.

Přínos práce spočívá ve využití výsledků při řešení konkrétních aplikací. Nejvhodnější způsob zajištění invariancí záleží na konkrétních požadavcích řešené aplikace, ale obecně nejlépe se jeví kombinace automatické volby bodu počátku popisu pro zajištění invariance vůči volbě bodu počátku popisu a rotace objektu pro zajištění invariance vůči rotaci.

Ze zjištěných výsledků je zcela zřejmé, že použití deformačních gramatik k rozpoznávání náhodně deformovaných objektů je možné.

10 LITERATURA

[1] Aho, Sethi, Ullman, Addison-Wesley: Compilers: Principles, Techniques, and Tools, 1986. ISBN 0-201-10088-6.

[2] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. Compilers: Principles, Techniques, and Tools Addison--Wesley, 1986.

[3] Deza, E.; Deza, M. (2006), Dictionary of Distances, Elsevier, ISBN 0444520872.

[4] Dick Grune, Ceriel J.H. Jacobs: Parsing Techniques, VU University Amsterdam, Amsterdam, The Netherlands. Originally published by Ellis Horwood, Chichester, England, 1990; ISBN-10: 038720248X, ISBN-13: 978-0387202488, dostupné z ftp://ftp.cs.vu.nl/pub/dick/PTAPG_1st_Edition/BookBody.pdf.

[5] Earley, J. (1968) An Efficient Context-Free Parsing Algorithm. PhD Thesis, Carnegie-Mellon University.

[6] Earley, J. (1970) An efficient context-free parsing algorithm. Commun. ACM, 13, 94–102.

[7] Gonzalez, Rafael C. & Woods, Richard E.: Thresholding. In Digital Image Processing, pp. 595–611. Pearson Education. ISBN 81-7808-629-8, 2002.

[8] Hlaváč, V., Šonka, M.: Počítačové vidění. Grada, Praha, 1993.

[9] Hopcroft, J. E., Ullman, J. D.: Formálne jazyky a automaty. Alfa, Bratislava, 1978.

[10] Chapman, S.: String Metrics, University of Sheffield, 2005. dostupné z: http://www.dcs.shef.ac.uk/~sam/stringmetrics.html.

[11] John Aycock, R. Nigel Horspool, „Practical Earley Parsing“, The Computer Journal, Vol. 45, No. 6, 2002, British Computer Society 2002, Pages 620-630.

[12] Levenshtein.net, dostupné z: http://www.levenshtein.net/.

[13] Luong Chi Mai, Introduction to Computer Vision and Image Processing E-book, Department of Pattern Recognition and Knowledge Engineering, Institute of Information Technology, Hanoi, Vietnam, dostupné z: http://eisc.univalle.edu.co/materias/Procesamiento_De_Imagenes_Digitales/Libro%20de%20Ayuda/Introduction%20to%20Image%20Processing%20and%20Computer%20Vision.pdf.

[14] M. Bouckaert, A. Pirotte, M. Snelling, “Efficient parsing algorithms for general context-free parsers”, Inform. Sci., vol. 8, no. 1, p. 1-26, Jan 1975.

[15] Mařík, V., Štepánková, O., Lažanský, J. a kolektiv: Umělá inteligence, část 4, Academia, vydání první, Praha, 2003, ISBN 80-200-1044-0.

[16] Masaru Tomita, Efficient parsing for natural language, Kluwer Academic Publishers, Boston, p. 201, 1986.

[17] Minařík, M., Šťastný, J., Popelka, O.: A Brief Introduction to Recognition of Deformed Objects, International Conference on Soft Computing Applied in Computer and Economic Environment ICSC, Kunovice, 2008, pp.55-64, ISBN 978-80-7314-134-9.

[18] Minařík, M., Šťastný, J.: Recognition Of Randomly Deformed Objects, 14th International Conference on Soft Computing, Mendel 2008, June 18-20, Brno, Czech Republic, pp.275-280, ISBN 978-80-214-3675-6.

[19] Minařík, M.: Strukturální metody identifikace objektů, FSI Junior konference 2006, leden 2007, Brno. s. 78-89. ISBN: 80-214-3364-9.

[20] Moore, Robert C.: Removing Left Recursion from Context-Free Grammars. 6th Applied Natural Language Processing Conference: 249-255. May 2000, dostupné z http://acl.ldc.upenn.edu/A/A00/A00-2033.pdf.

[21] Philippe McLean & R. Nigel Horspool, A Faster Earley Parser, Dept. of Computer Science, University of Victoria, Victoria, BC, Canada V8W 3P6, dostupné z: http://webhome.cs.uvic.ca/~nigelh/Publications/fastEarley.pdf.

[22] Resolving the General Dangling Else/If-Else Ambiguity, dostupné z: http://www.parsifalsoft.com/ifelse.html.

[23] Richard W. Hamming. Error Detecting and Error Correcting Codes, Bell System Technical Journal 26(2):147-160, 1950.

[24] Rybička, J.: Úvod do teorie formálních jazyků, učební pomůcka, nakladatelství Konvoj, Brno 1999.

[25] S.H. Unger, "A global parser for context-free phrase structure grammars", Commun. ACM, vol. 11, no. 4, p. 240-247, April 1968.

[26] S.L. Graham, M.A. Harrison, W.L. Ruzzo, "An improved context-free recognizer", ACM Trans. Prog. Lang. Syst., vol. 2, no. 3, p. 415-462, July 1980.

[27] Seifert, S.: Ein Earley-Parser fur Zeichenketten Erzeugende Hypergraphgrammatiken, Studienarbeit im Fach Informatik, 2004, dostupné z http://www2.informatik.uni-erlangen.de/Lehre/SA-DA/download/SA-seifert.pdf.

[28] Sipser, Michael: Introduction to the Theory of Computation (1st ed.), IPS, p. 99, ISBN 0-534-94728-X, 1997.

[29] Šťastný, J., Minařík, M. Object Recognition by Means of New Algorithms. ICSC - International Conference on Soft Computing Applied in Computer and Economic Enviroment. Kunovice: Evropský polytechnický institut, Kunovice, 2006. s. 99-104. ISBN: 80-7314-084-5.

[30] Šťastný, J., Škorpil, V.: Analysis of Methods for Edge Detection. International journal Communications, 2003, ISSN 0018-2028, 19 pp.

[31] Šťastný, J., Škorpil, V.: Filtering Methods for Picture Processing, International Conference TELECOMMUNICATIONS AND SIGNAL PROCESSING- TSP 99, Brno, 1999.

[32] Šťastný, J.: Netradiční metody a algoritmy pro rozpoznávání objektů technologické scény, zkrácená verze habilitační práce FSI VUT, Brno, 2005, ISBN 80-214-3117-2.

[33] Šťastný, J.; Minařík, M. A Brief Introduction to Image Pre-Processing for Object Recognition. ICSC - International Conference on Soft Computing Applied in Computer and Economic Enviroment. Kunovice: Evropský polytechnický institut, Kunovice, 2007. s. 55-64. ISBN: 80-7314-084-5.

[34] The Math Explorers' Club, Euclidean Distance, dostupné z: http://www.math.cornell.edu/~mec/Summer2008/Jones/euclid.htm.

[35] Železný, M.: Strukturální metody rozpoznávání – přednášky, katedra kybernetiky Fakulty aplikovaných věd, Západočeská univerzita v Plzni, dostupné z: http://artin.zcu.cz/courses/smr/Smr-031021.pdf.

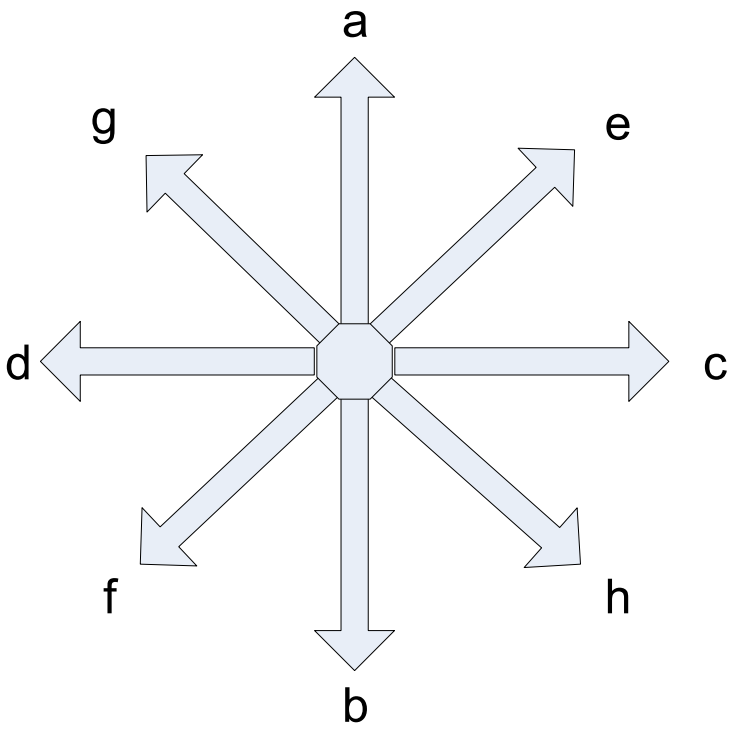
[36] Železný, M.: Zpracování digitalizovaného obrazu – přednášky, katedra kybernetiky Fakulty aplikovaných věd, Západočeská univerzita v Plzni, dostupné z: http://artin.zcu.cz/courses/zdo/ZDO_aktual_060217.pdf.

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM GRAFŮ

PŘÍLOHA A – KLASICKÁ PRIMITIVA



PŘÍLOHA B – DIFERENCIÁLNÍ PRIMITIVA